水下传感器的三维网络部署研究

摘要

水下无线传感器网络（Underwater Wireless Sensor Networks, UWSN）[1][2]是由多个低能耗、通信距离受限的传感器节点组成的水下检测系统，其可采集指定区域内的信息并进行收集整理。水下无线传感器网络目前广泛应用于海洋资源开发、水下环境监测等方面。由于水下无线传感器网络的广阔应用前景，已引起了各界的极大关注。其中，节点所传输的信息的质量又与节点部署策略息息相关。因此，如何合理得对节点位置进行部署成了水下无线传感器网络设计的核心。

问题一中，我们首先将拥有探测半径的传感器节点视为一个球，根据球最大内接立方体的方式对海域进行传感器节点的部署。随后发现在二维情况下[3][4]圆对正方形进行填充时，以圆的最大内接正方形填充并不一定是最佳方案，于是我们将海域划分为 $100\times100\times100$ 的网格[5][6]，将对海域空间的覆盖考虑成对网格交点的覆盖，以最初的节点坐标为基准，采取节点坐标在局部范围内随机的方式进行迭代，在保障网格覆盖率的同时，又通过各节点的位置信息和通信限制条件，建立每一个节点的邻接表，并使用 BFS 保障网络可通信，即网络节点至少满足可形成生成树[7]。我们在一定时间将得到大量可行解，并根据这些可行解建立了多目标优化模型，其中目标函数为传感器节点数和覆盖率。最终解得 100%覆盖海域时，最少需要 125 个传感器节点；90%覆盖海域时，最少需要 79 个传感器节点；80%覆盖海域时，最少需要 64 个传感节点。

问题二中，我们使用了一个多叉树模型[7]，其以基站为树根，以与基站可通信的所有节点作为第一层节点。我们采用 BFS 的方式得到整棵树各层节点[6] 以建立该树形模型，为了平衡各节点网络压力，我们引入了期望子节点数的概念并利用最小堆不断维护建立了初始的多叉树。由于期望子节点数存在相近节点侵占对方子节点的问题，我们使用了自创的最大最小堆联合维护算法微调树链结构，最终获得在题目一模型下的具有最佳通信效率的树形结构。通过节点在在树上的深度和其被固定窗口大小的 BFS 访问的时间之间的关系，得到所有节点的信号分九个批次进行发送。

问题三中，依据问题二建立的树形结构模型，可得知网络传输压力主要集中在第一层的节点上，由于高度为 1 的树节点的子节点数量不均，若某个位于第一层的节点失效，则可以认为该树形传输结构无法覆盖整个海域。因此我们要做的是考虑最先失效的节点所能保持通信的时间。所有节点一直在实时地进行数据传递，对于以第一层节点为根的每一颗子树，当子树所有的节点信号都发给基站时，则需要从树根开始重发新产生的数据。考虑节点发送信号和接收信号的平均时间和功率，再结合每个第一层元素的子树大小，建立节点数量与节点单位时间功率的关系式，对该关系式求导，得出其与节点数量是否成正相关，由此可以得到系统最先失效的节点。该节点所可通信的时间即整个网络的可通信时间为 1486.06s，考虑到有两个节点都可以运行该时间，于是我们可以在其位置各增加一个节点。

关键词：图，树，堆，邻接表，网格遍历，BFS，多目标优化，树链，贪心

一．问题重述

1.1 问题背景

水下无线传感器网络是通过将大量传感器节点部署在不同水域内并采用声波作为通信介质进行传输的水下网络数据连接。它在开发海洋资源、水下数据采集以及环境检测等方面都有很广泛的应用。由于水下环境的特殊性和节点成本高、网络耗能大等原因，如何合理得布置节点以节省节点资源并减少网络能耗成了研究的首要问题。

1.2 问题要求

根据上述题目背景，题目要求建立数学模型讨论以下问题：

1. 用以普通节点为球心、探测半径为 20m 的球对$100\text{m} \times 100\text{m} \times 100\text{m}$实现全覆盖，同时满足采集获得信息可传输至基站节点，至少需要多少个普通节点以及它们的部署位置?
2. 当覆盖率为 90%和 80%时，求节点数量和位置的变化。
3. 根据普通节点与普通节点以及基站节点间的信息发送、接受条件，为整个网络规划信息发送方案，网络开始后，每个传感器何时将监测获取的信息发送至基站?
4. 满足覆盖性、连通性与节能性条件，所得到的节点部署方案在多长时间实现整个水域的全面检测？
5. 若增加普通节点数量或者调整普通节点位置，该如何调整？

二．模型假设

1. 假设正方体水域介质均匀分布，即所有普通节点的信号在水中的传播速度一致。
2. 不考虑水流速度、水下温度、不同深度压强或水下生物等环境因素以及信号折射、反射等现象对信号传输带来的影响。
3. 所有节点的规格参数相同，不考虑节点个体差异性对部署位置选择的影响。
4. 三维空间坐标系的原点为基站位置，并以空间竖直向上作为 y 轴正方向，向右为 x 轴正方向，向外为 z 轴正方向。
5. 传感器节点在发送节点数据的同时可以接收其他节点给他发来的数据。
6. 传感器节点可以暂时保存其他节点对其发送的数据。

三．符号说明

符号	符号含义
H_i	立方体i的边长
$\mathrm{R_i}$	球i的半径
$R_{B.S}$	基站节点的感知传输半径
$R_{\exp}$	普通节点的探测半径
R_{tran}	普通节点的传输半径
$\mathrm{O_j}$	球j

C_i	立方体i
Se	立方体海域
Q_i	正方形i
S_i	正方形i的边长
$\Re_i$	水下传感器普通节点i
Ω	所需最少节点数
X_i	点i的x轴坐标
Y_i	点i的y轴坐标
Z_i	点i的z轴坐标
$\aleph_{ij}$	节点i和 j 是否连通
R_{mov}	普通节点可自由运动半径
ω	海域探测覆盖率
$\mathbb{N}$	期望子节点数
P	功率
T	耗时
E	能量

四．问题分析

4.1 问题一分析

问题一要求我们部署普通传感器节点对一片体积为$100m\times100m\times100m$的海域实现全覆盖，同时满足每一个节点采集的信息能够传输至基站节点。要求我们深入讨论，若将覆盖区域分别改为原先的90%和80%，节点的位置和数量会发生什么改变。

我们将传感器节点在海域中的探测空间抽象为几何球体，将海域抽象成一个$100m\times100m\times100m$的立方体，从球的最大内接立方体对空间立方体的填充出发，再通过对二维平面内圆对正方形填充规律的探查[3][4]，发现球内接立方体式的填充方案并不一定是最优解。所以我们将海域划分为$100\times100\times100$的网格，并定义网格连线之间的交点为基点，那么我们将得到$101\times101\times101$个基点，利用上述球内接最大立方体式填充得到的125个节点坐标作为初始值，模拟微观粒子学上分子无规则运动的特征，使每一个节点在某个限定范围内进行随机震荡，在保证覆盖率和满足生成树的基础上得到一系列局部最优解。最终，我们从一系列传感器节点数量已达最优的解中选取覆盖基点数最多的解作为最终答案。

4.2 问题二分析

1) 每个普通节点在同一时间只能向一个节点发送数据
2) 每个普通节点只能在同一时间接收来自另一个普通节点的数据
3) 基站节点可以同时接收多个普通节点发送的数据
4) 每个节点仅能发送包含一个节点监控区域信息的信息量

问题二基于以上四点假设，要求我们根据问题一的节点部署方案，求解每一个节点何时将数据发送给基站。

问题二中，我们利用传感器节点在可通信约束条件下得到的邻接表建立出整张图的模型。因为基站可以同时接收多个节点发送的信息，所以，我们考虑到与其可通信节点应作为其他节点的转发器，且尽可能具有相近的压力。此外，我们可以明确地知道，这些节点之间的互相通信除了增加网络压力，并不会带来任何有用的结果。从递归的角度进行思考，对于这一批节点的子节点，他们之间的互相通信也只会降低网络信息传输效率。那么我们可以选择删除图上诸如此类没用的边，然后可从原始网络图中抽象出一张子图。该子图为一棵多叉树，它以基站为树根，且以与基站可通信的所有节点作为树的第一层节点。我们可采用 BFS 染色法[6]求生成树的方式来得到整棵树的各层节点。

为了平衡各节点的通信压力，我们引入了期望子节点数的概念并利用最小堆[7]进行维护建得初始的多叉树，期望子节点数为该节点可能的子节点数量，文中我们会给出严谨的数学定义。位居堆顶的元素，它的期望子节点最少，为防止他的可能子节点被其他同层节点优先获得，我们让其最先择子。每次有一个节点择子后，整个体系周围节点的期望子节点数都会受到影响，所以我们要对期望表进行更新，并一直用最小堆进行维护，保障堆顶元素的期望子节点最少。

但期望子节点有可能会存在两个期望数相同的节点其中某个节点侵占另一个节点子的情况，对于期望子节点数的这个不足，我们使用了自创的最大最小堆联合维护算法微调树链结构，将部分被侵占的子节点所在树链归还给被侵占节点。

最终获得在题目一模型下的具有最佳通信效率的树形结构。

由于最后的转发器只存在和传感器直接可通信的节点，我们就可根据此类节点数量，设置固定窗口大小的 BFS，得到每一个节点被访问时的时间，该时间就是该节点的信息到达基站所需的时间。假定初始时刻为 0，那么其发送时刻就可用所需时间与节点所在树上的深度来求得。

我们通过程序(9.1.1 问题二代码一)解得树的期望结构，并在此基础上利用(9.1.1 问题二代码二)微调树链。对于最终的模型，我们可使用(9.1.1 问题二代码三)求得各节点发送信息的时刻

4.3 问题三分析

在问题三中，每个节点拥有初始能量 30J，当节点能量低于 0.1J 时，则该节点失效，对于问题 2 建立的树形结构模型，网络传输压力主要集中在第一层的节点，由于第一层网络节点的子节点数量并不均匀，其中必然会有一个或者若干个节点的能力率先低于 0.1J，所以，对于该模型，如果某个第一层的节点失效，我们可以认为该树形传输结构便无法覆盖整个海域。所以，我们要做的是考虑最先失效的节点他所能维持的时间。

因为所有节点一直在实时的进行数据的传递，对于以第一层节点为根的每一颗子树，当子树所有的节点的信号都发给基站时，则需要根从头开始重发新产生的数据。

考虑节点发送信号和接收信号的平均时间和功率，再结合每个第一层元素的子树大小，建立节点数量与节点单位时间功率的关系式，对该关系求导，得到其与节点数量的

是否成正相关，由此可以得到系统最先失效的节点。此时，网络无法保障 100%的覆盖率，所以我们只要在该点位置增加一个节点，就可以使该点压力减半，我们可以解得所有第一层节点的可通信时间，选择增加合适的节点使得系统的利用率尽量增大。

五．模型建立与求解

5.1 问题一：不同覆盖程度条件下水下传感器节点的数量及位置分布

为了明确空间模型，空间坐标系模型和网格划分模型如下图所示：

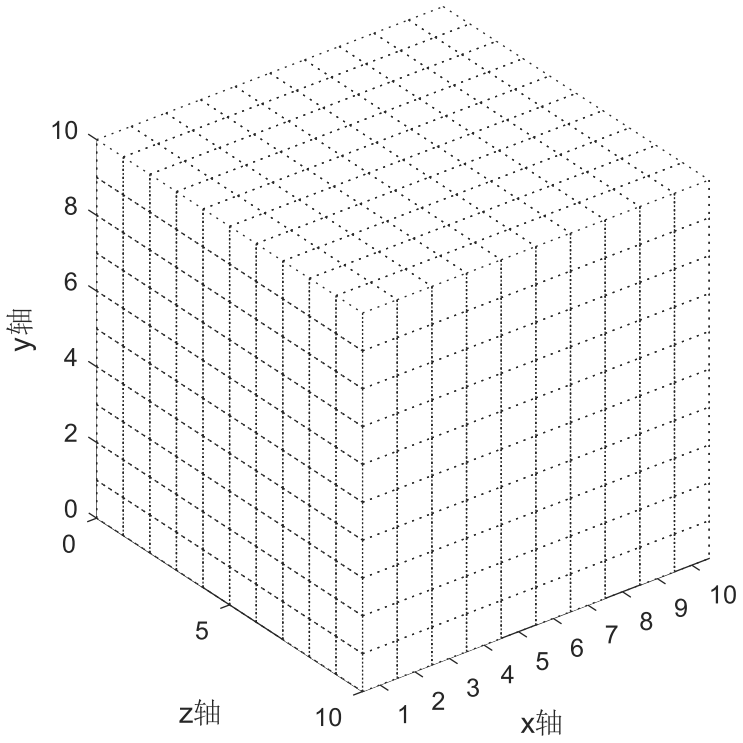

图 1 空间坐标系及网格划分模型

图 1 中坐标系仅表示实际坐标系方向，且网格数量仅供参考。

5.1.1. 球的最大内接立方体对空间立方体的填充模型

对于任意给定空间球O_α，对于其最大内接立方体C_α，立方体边长有如下公式：

$$H_i = \frac{2R_\alpha}{\sqrt{3}} \tag{1}$$

每一个球都可以视为一个边长为Hi的立方体，使用立方体C_i对海域S进行一个无空隙的填充。由于海域边长$H_S = 100m$，所以我们填充海域所需立方体个数：

$$N_S = \left(\left\lceil \frac{H_S}{H_i} \right\rceil \right)^3 \tag{2}$$

将得到如下所示空间排列模型：

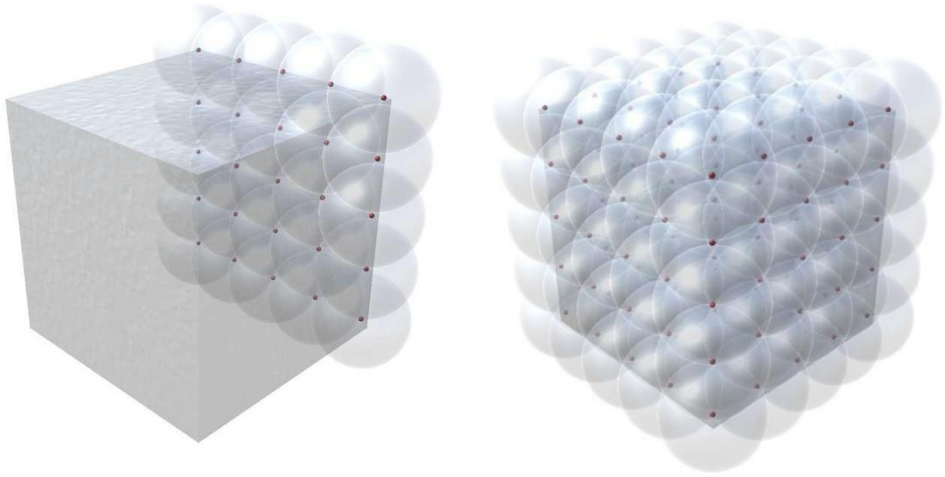

图 2 单层球最大内接立方体填充模型　　图 3 球最大内接立方体全填充模型

图 2 和图 3 中半透明深色立方体为海域Se，半透明圆球为传感器节点的探测空间，透明球中的实心红点为传感器节点所在空间位置。

观察模型所示填充方案，我们发现，空间内探测区域的重叠十分密集，于是猜想，球最大内接立方体式填充可能并不是当前空间的最佳填充方案。

我们从三维空间回到 2 维空间对圆填充正方形问题进行分析。

5.1.2. 二维平面内圆对正方形填充分析

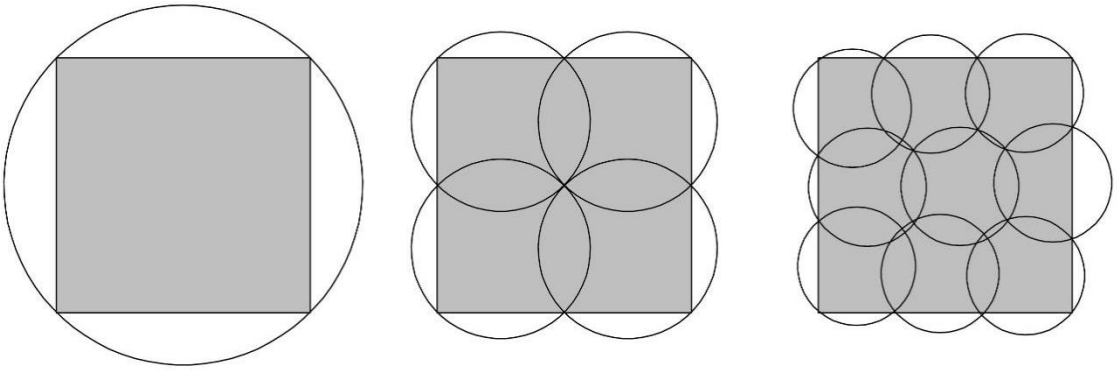

图 4 二维单位单位圆内接正方形最大填充图

图 4 中，所有的圆都是单位圆，每幅图对应数量的单位圆在图示情况下将得到覆盖面积最大的正方形$Q_{\max}$。

我们假设$\Im^2$个单位圆所能填充的最大正方形边长

$$S_{\max}=\sqrt{2}\Im \tag{3}$$

根据公式(3)我们应有$S_{Q_1}=1.414$，$S_{Q_2}=2.828$，$S_{Q_3}=4.242$；根据资料[3][4]我们可以得到$S_{Q_1}'=1.414$，$S_{Q_2}'=2.828$，$S_{Q_3}'=4.335$，存在$S_{Q_3}\neq S_{Q_3}'$，所以假设不成立，式(3)不正确。

由此可知，对于三维空间，以球的最大内接立方体对海域Se进行完全覆盖的填充并不一定是所用传感器节点最少的填充方式。

5.1.3 三维空间约束分析

上文已经提出了球的最大内接立方体对空间立方体的填充模型，要满足将采集获得的信息传输至基站节点 $B.S$ 的要求，我们在下文根据普通节点 $\Re_i$ 的探测半径 $R_{\exp}$ 与传输半径 R_{tran}、基站节点 $B.S$ 的感知传输半径 $R_{B.S}$、多跳传输形式等条件约束，对普通节点 $\Re_i$ 的部署位置进行分析。

（1）普通节点的固有探测半径对普通节点部署位置的约束

探测半径 $R_{\exp}$ 的含义为在以普通节点 $\Re_i$ 为球心，固有探测半径 $R_{\exp}$ 为半径的球面覆盖的海域 Se 内，海洋信息均可被探测。由此，该约束用数学语言可描述为：正方体海域 Se 内的任一基点 k 与海域内任意普通节点 $\Re_i$ 间的距离的最小值小于等于普通节点固有的探测半径 $R_{\exp}$，即：

$$\min\sqrt{(X_k-X_{\Re_i})^2+(Y_k-Y_{\Re_i})^2+(Z_k-Z_{\Re_i})^2}\le R_{\exp},\ (k,i\in \mathrm{N}^*) \tag{4}$$

（2）普通节点的固有传输半径对普通节点部署位置的约束

传输半径 R_{tran} 的含义为只有任一普通节点 $\Re_i$ 在以另一普通节点 $\Re_j(j\neq i)$ 为球心，固有传输半径 R_{tran} 为半径的球面覆盖的海域 Se 内，两节点方可产生通信业务。由此，该约束用数学语言可描述为：正方体海域 Se 内的任一普通节点 $\Re_i$ 与海域内其余普通节点 $\Re_j(j\neq i)$ 间的距离的最小值小于等于普通节点固有的传输半径 R_{tran}，即：

$$\min\sqrt{(X_{\Re_i}-X_{\Re_j})^2+(Y_{\Re_i}-Y_{\Re_j})^2+(Z_{\Re_i}-Z_{\Re_j})^2}\le R_{\text{tran}},\ (i,j\in \mathrm{N}^*) \tag{5}$$

（3）基站节点的感知传输半径对普通节点部署位置的约束

感知传输半径 $R_{B.S}$ 的含义为只有在以基站节点 $B.S$ 为球心，感知传输半径 $R_{B.S}$ 为半径的球面覆盖的海域 Se 内的普通节点 $\Re_i$，才能将信号传输给基站节点 $B.S$。由此，该约束用数学语言可描述为：正方体海域 Se 内的基站节点 $B.S$ 与海域内任一普通节点 $\Re_i$ 间的距离的最小值小于等于基站节点的感知传输半径 $R_{B.S}$，即：

$$\min\sqrt{(X_{B.S}-X_{\Re_i})^2+(Y_{B.S}-Y_{\Re_i})^2+(Z_{B.S}-Z_{\Re_i})^2}\le R_{B.S},\ (i\in \mathrm{N}^*) \tag{6}$$

（4）信息多跳传输形式对普通节点部署位置的约束

信息多跳传输形式的含义为任一普通节点 $\Re_i$ 所采集的海洋信息，均可在任意两个可产生通信业务的节点间传输，并且信息通过不断传输，最终能够传达到基站节点 $B.S$。由此，采用“0-1 规划”，若两节点 $\Re_i$ 与 $\Re_j(j\neq i)$ 之间可产生通信业务(即满足以下两个约束，（1）普通节点的固有传输半径对普通节点部署位置的约束和（2）基站节点的感知传输半径对普通节点部署位置的约束)，则节点连通状态 $\aleph_{ij}$ 为 1，否则，$\aleph_{ij}$ 为 0。 若从任一普通节点 $\Re_i$ 至基站节点 $B.S$ 间任意两节点的连通状态为 1，则该普通节点的信息传输状态为 1，即 $\prod\aleph_{ij}=1$，否则为 0。分析题目可知，所有采集到的信息均可传输至基站节点，则所有普通节点的信息传输状态均为 1，即 $\prod\prod\aleph_{ij}=1$。

5.1.4 传感器节点坐标的随机震荡

对于普通水下传感器节点 $\Re_i(i\in[1,\Omega])$，有：

$$\exists(R_{mov}\le 2R_{\exp}),\sum_{i=1}^{\Omega}(\Re_i\in\begin{Bmatrix}X_{\Re}\le -R_{\exp}-\frac{H_{Se}}{2}\\ X_{\Re}\ge \frac{H_{Se}}{2}+R_{\exp}\\ Y_{\Re}\le -R_{\exp}-H_{Se}\\ Y_{\Re}\ge R_{\exp}\\ Z_{\Re}\le -R_{\exp}-\frac{H_{Se}}{2}\\ Z_{\Re}\ge \frac{H_{Se}}{2}+R_{\exp}\end{Bmatrix})\ge 98 \tag{7}$$

即必然存在 R_{mov} 使得 Se 中的的传感器数量在27以内。那么必有：

$$\alpha\le\frac{36\pi R_{\exp}^{\ 3}}{H_{Se}^{\ 3}}\le 90.43\% \tag{8}$$

但在实际中，27个探测器节点探测空间必然存在大量重叠部分，若考虑所有27个节点按照不重叠的方式对海域 Se 进行探测，则将浪费约 $12\pi R_{\exp}^{\ 3}$ 的空间，此时：

$$\alpha\le\frac{24\pi R_{\exp}^{\ 3}}{H_{Se}^{\ 3}}\le 60.28\% \tag{9}$$

此时覆盖率小于问题一中的最少要求80%，所以我们可以 $2R_{\exp}$ 作为随机震荡半径。在每一次震荡中，如果我们得出了一种所需节点数比当前剩余节点数还少的解的情况，那么我们就可以从我们的节点集合中剔除所有满足式(7)的节点，因为该节点必然不被最优化结果模型需要。剔除多余点不仅保证了下一次震荡所得解必然优于或等于当前解，还大大缩小了解空间。随机震荡将以下图所示趋于最优解。

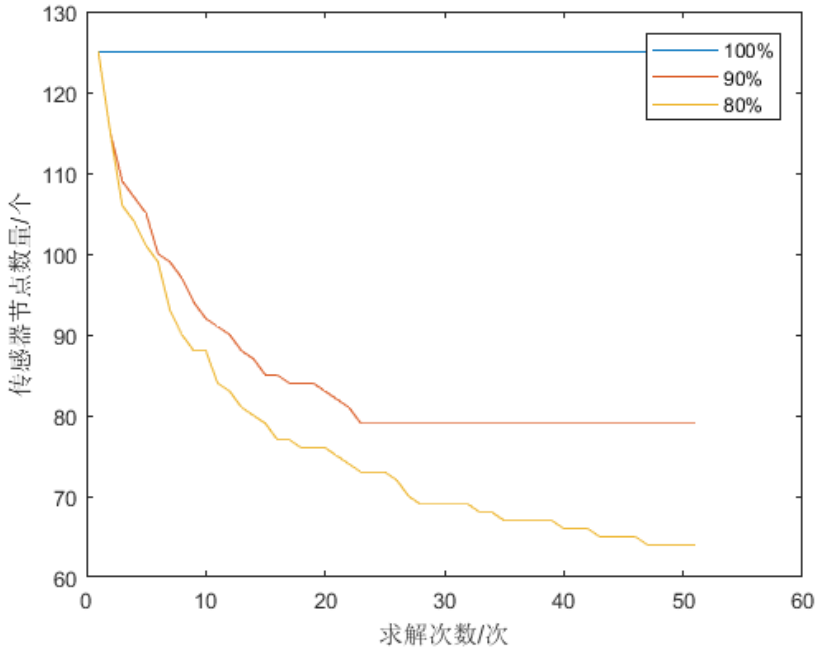

图 5 传感器节点数量震荡趋势图

5.1.5 网格遍历保障覆盖率[5][6]

基于图 1 所示的网格划分模型我们得到101×101×101个基点，我们采用以基点覆盖率代替海域覆盖率的方式，那么本问题即解在传感器节点满足式(4)(5)(6)所述空间约束的条件下，基点覆盖率达到要求时所需要的尽可能少的节点数。

对于每一次震荡之后的节点集合坐标，我们进行网格遍历，检验其网格覆盖率。对于没达到要求覆盖率的情况，则恢复各节点坐标为震荡前的状态再次震荡；对于满足覆盖率的情况，则再次检验其所有节点是否可以进行通信，即所有节点能否连通形成一棵生成树。

5.1.6 基于广度优先搜索（Breadth-First Search）的可通信保障

对于所有节点组成的集合，如果其中任意两个点i 和 j可通信，那么他们的联通状态 $\aleph_{ij}$ 则为 1，否则为 0，由此建立邻接表，如果可以连通形成生成树，那么必然可以从基站 *B.S* 出发，根据邻接表遍历状态遍历所有节点。

如果该情况下的节点集合无法形成生成树，那么我们重新震荡节点，重新生成节点坐标。

5.1.7 对于节点数、覆盖率的多目标选择

在空间震荡失败若干次仍然没有新解产生时，我们可以认为当前已经获得了满足覆盖要求的最少节点数量。通过程序（9.1.1 问题一代码一）获得数据如 9.2.1 所示，对于所需节点数相同的解，我们优先选择能覆盖基点较多解。

最终，对于问题一的三种覆盖率，我们得到的最优解如下表所示（完整表格见附录 9.2.1）：

表 1 覆盖率为 100%时，节点数量及位置坐标表

	X	Y	Z		X	Y	Z		X	Y	Z		X	Y	Z		X	Y	Z
1	-38.453	-88.453	-38.453	26	-15.359	-88.453	-38.453	51	7.735	-88.453	-38.453	76	30.829	-88.453	-38.453	101	38.453	-88.453	-38.453
2	-38.453	-88.453	-15.359	27	-15.359	-88.453	-15.359	52	7.735	-88.453	-15.359	77	30.829	-88.453	-15.359	102	38.453	-88.453	-15.359
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	..
24	-38.453	-11.547	30.829	49	-15.359	-11.547	30.829	74	7.735	-11.547	30.829	99	30.829	-11.547	30.829	124	38.453	-11.547	30.829
25	-38.453	-11.547	38.453	50	-15.359	-11.547	38.453	75	7.735	-11.547	38.453	100	30.829	-11.547	38.453	125	38.453	-11.547	38.453

表 2 覆盖率为 90%时，节点数量及位置坐标表

	X	Y	Z		X	Y	Z		X	Y	Z		X	Y	Z
1	-38.453	-91.453	-38.453	21	-27.359	-99.453	21.735	41	3.735	-88.453	5.735	61	17.829	-92.453	3.735
2	-19.453	-72.453	3.641	22	-34.359	-69.453	23.829	42	25.735	-88.453	23.829	62	27.829	-70.453	37.829
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
19	-30.359	-78.453	-45.453	39	5.735	-90.453	-24.453	59	35.829	-82.453	-40.453	79	41.923	-12.171	-8.265
20	-5.359	-78.453	-15.359	40	-10.265	-81.453	-13.359	60	30.829	-98.453	-26.359				

表 3 覆盖率为 80%时，节点数量及位置坐标表

	X	Y	Z		X	Y	Z		X	Y	Z		X	Y	Z
1	-46.453	-83.453	-38.453	17	-11.359	-88.453	-23.453	33	21.735	-88.453	-19.453	49	34.829	-79.453	-22.453
2	-20.453	-92.453	-16.359	18	-15.359	-70.453	-32.359	34	-2.265	-88.453	-6.359	50	30.829	-88.453	-30.359
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
15	-19.453	-5.171	-8.265	31	-16.359	-26.171	14.735	47	-10.265	-5.171	-1.265	63	17.829	-30.171	12.735
16	-24.453	-37.171	42.829	32	-32.359	-2.171	30.829	48	4.735	-24.171	41.829	64	24.829	-8.171	20.829

5.1.8 程序流程图

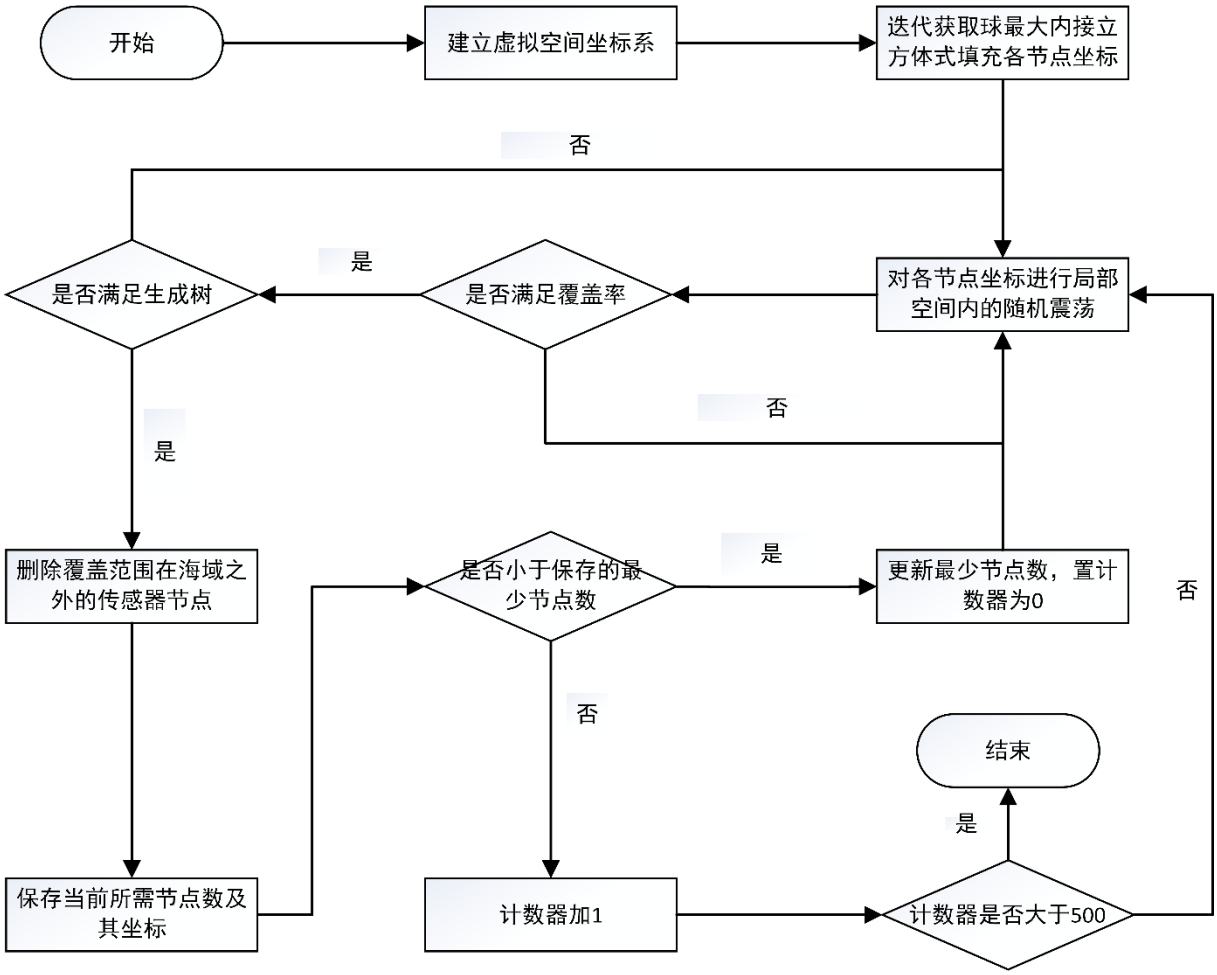

图 6 问题一程序流程图

5.2 问题二 网络信息发送方案

由于基站可以同时接收来自多个节点的信息，根据贪心策略，应让可以和基站 *B.S* 通信的节点 $\Re_{cnc}$ 尽可能多的发送数据，同时为了分散网络压力，我们尽可能让 $\Re_{cnc}$ 均匀管理与其可通信的子节点，那么，这个网络通信图的问题我们可以抽象成一个多叉树问题。以B.S作为树根，所有 $\Re_{cnc}$ 即为树的第一层节点。

为了方便描述，各节点索引即表 1 所示。

5.2.1 利用 BFS 得到树结构各层节点

为了得到树每一层的节点，我们采用 BFS 算法依据邻接表进行图的搜索，并采取对以搜索节点进行染色的策略来保障我们搜索得到的是一颗树，最终得到各层节点信息如下表（其中，没有子的树节点已在表格中省略）：

表 4 多叉树各层节点

0	B.S
1	42 43 47 48 67 68 72 73 74 98
2	17 22 37 41 46 18 23 38 44 49 62 66 71 92 97 63 69 93 45 50 70 75 94 95 99 100 117 118 119 122 123 124
3	12 16 21 32 36 13 19 24 33 39 20 25 40 57 61 87 91 96 112 116 121 58 64 88 65 113 89 90 114 115 120 125
4	7 11 27 31 8 14 15 28 34 35 52 56 82 86 107 111 53 59 83 60 108 84 85 109 110
5	2 6 26 3 9 10 29 30 51 77 81 102 106 54 78 55 103 79 80 104 105
6	1 4 5 76 101

5.2.2 期望子节点数

为了使所得多叉树各节点能够均衡网络压力，我们引入期望子节点树的概念。将节点x的期望子节点数定义为：

$$\mathbb{N}_x = \mathrm{son}(x) + \sum_{i=0}^{size(adj\_list[x])-1} \frac{1}{acc\_brother\_count(adj\_list[x][i])} \tag{10}$$

该式中 $son(x)$ 为节点x已经存在的子节点的数量；$adj\_list[x]$ 为x节点的邻接表，$\mathrm{size}(adj\_list[x])$ 即为邻接表大小；acc_brother_count(adj_list[x][i]) 代表x的兄弟节点(包括自身)，能够根据邻接表到达邻接表存储节点 $adj\_list[x][i]$ 的数量。

对于此定义，解释为节点x的期望子节点数等于节点x当前已拥有子节点数 $son(x)$ 与其邻接表可到达的每一个节点 $adj\_list[x][i]$ 的邻接表中包含x兄弟节点的个数的倒数和。之所以取倒数是因为该节点拥有相同概率被其兄弟访问，所以x访问该节点的价值期望为：

$$\begin{aligned} E(x) &= 1 \times \frac{1}{acc\_brother\_count(adj\_listp[x][i])} \\ &= \frac{1}{acc\_brother\_count(adj\_listp[x][i])} \end{aligned} \tag{11}$$

5.2.3 利用最小堆维护多叉树以均衡网络压力[8]

在引入期望子节点数之后，我们可以利用最小堆依据期望子节点数维护各层的节点集合。位于堆顶的节点拥有优先择子权，且其择子对象仅为其下一层可通信节点。通过程序(9.1.1 问题二代码一)依据此策略建树，最终得到期望上较优的多叉树结构如下表所示：

表 5 第一层节点的子节点对应表

父节点	子节点	父节点	子节点	父节点	子节点
42	22 37	43	18 38 49	47	17 41 46
48	23 44	67	62 97	68	63
72	66 71	73	69 93	74	45 50 70 75 95 100
98	92 94 99 117 118 119 122 123 124				

表 6 第二层节点的子节点对应表

父节点	子节点	父节点	子节点	父节点	子节点
17	12	22	16	37	32
41	36	46	21	18	13
23	19	38	33 39	49	20
62	57	66	61	71	91
92	112	97	96 121	63	58
69	64	93	88	45	40
50	24 25	70	65	94	90
95	114 115	99	120	100	125
117	87	118	113	119	89
122	116				

表 7 第三层节点的子节点对应表

父节点	子节点	父节点	子节点	父节点	子节点
12	7	16	11	32	27
36	31	13	8	19	14
33	28	39	35	20	15
40	34	57	52 82	61	56
91	111	112	107	116	86
58	53	64	59	88	83
65	60	113	108	89	110
90	84	114	109	115	85

表 8 第四层节点的子节点对应表

父节点	子节点	父节点	子节点	父节点	子节点
7	2	11	6	27	26
8	3	14	9	15	10
28	29	35	30	52	77
56	51	86	81	107	102
111	106	53	54 78	83	103
60	55	84	105	85	104
109	80	110	79		

表 9 第五层节点的子节点对应表

父节点	子节点	父节点	子节点	父节点	子节点
2	1	3	4	29	5
51	76	77	101		

分析上述表格我们发现，树的第一层元素存在子节点分配不均的问题，结合第一层节点的邻接表，分配不均问题可能由以下两个因素造成：

1) 节点空间位置不同带来的各节点邻接表大小不同

2) 在利用期望子节点数该概念进行择子的时候可能会发生同级节点侵占现象，即两个期望子节点数相同的节点，其中一个先择子的时候侵占了另一个节点的择子空间。

1)中所述问题表明邻接表和传感器节点坐标有关，我们所求得的各节点坐标已经是问题一中在满足全覆盖条件下的最优解，因此可忽视 1)。2）中，为了消除该情况带来的影响，我们使用了最大堆和最小堆联合维护的算法微调以上结果，以得到最优解。

5.2.4 最大堆和最小堆联合维护算法微调结果得到最优解

根据树第一层元素的邻接表，我们所得到的树模型中，存在个别多子节点侵占了少子节点子的情况。也就是说，如果能将多子节点侵占的节点归还给少子节点，就可尽可能均匀网络压力，提高网络通信效率。于是我们采用了一个最大堆和最小堆联合维护的算法，算法介绍如下：

堆根据子节点的数量进行排序，为了简化算法的描述，我们考虑最大堆和最小堆为一个恒有序的线性表，以左侧作为堆顶，并按照第一层节点各自孩子数从多到少依次给以编号 1 到 10，初始堆状态如下：

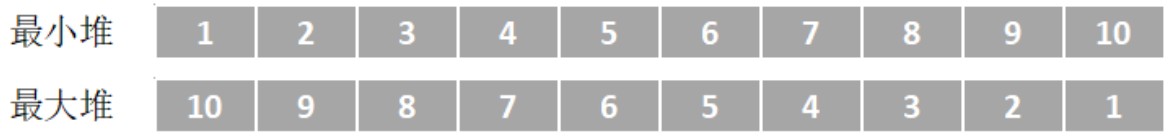

图 7 初始最大堆最小堆状态图

对于每次所弹出的最大堆中的堆顶元素 $\Re_i$，我们枚举其子节点。对于每一个子节点 $\Re_{\Re_i}$，我们从弹出的最小堆的堆顶元素 $\Re_j$ 开始，查询其邻接表 $adj\_list[\Re_j]$。若邻接表中存在 $\Re_{\Re_i}$，那么我们就在 $\Re_i$ 的子节点中删除 $\Re_{\Re_i}$，并把该子节点叠加给 $\Re_j$，同时，结束此次遍历，整合两个堆的元素后恢复为初始状态。若邻接表中不存在 $\Re_{\Re_i}$，则舍弃该最小堆堆顶元素，继续弹出最小堆堆顶元素做查询操作，直到最小堆为空。其中，堆恢复算法入下图所示：

图 8 堆恢复算法示意图

假设当前最大堆弹出元素为 10，最小堆弹出元素为 4，在 $adj\_list[4]$ 中我们可找到 10 的某一个子节点 $\Re_{10}$，然后我们将图示红色部分压入最小堆。之后，不断弹出最大堆蓝色元素，将剩下的绿色元素也压入最小堆，最后将最小堆拷贝给最大堆，得以恢复两

个堆的元素。此时，两个堆中红色所示元素的子节点数量已经得到更新，将会在堆中自动重新排序。下次执行上述算法时，依然可以保障最大堆中堆顶元素子节点数最多。

如果最大堆中堆顶元素 $\Re_i$ 的子节点在最小堆所有元素的邻接表 $adj\_list[\Re_j]$ 中没有出现，那么我们可以认为该堆顶元素子节点情况已达最优，因此此元素不必放回两个堆中，而是将其暂存于一个临时的容器。同时最小堆的恢复采用拷贝最大堆的方式，当最大堆和最小堆都为空时，我们将在临时容器中得到初始堆中的所有元素，且每一个元素的孩子分布已经达最优。

经过该算法的处理，我们得到第一层元素各节点子节点的最终情况。如下表所示：

表 10 第一层节点的子节点最终对应表

父节点	子节点	父节点	子节点	父节点	子节点
42	22 37	43	18 38 49	47	17 41 46
48	23 44	67	62 97	68	63
72	66 71 92	73	69 93	74	45 50 70 75 94 95 100
98	99 117 118 119 122 123 124				

该算法流程图如下所示：

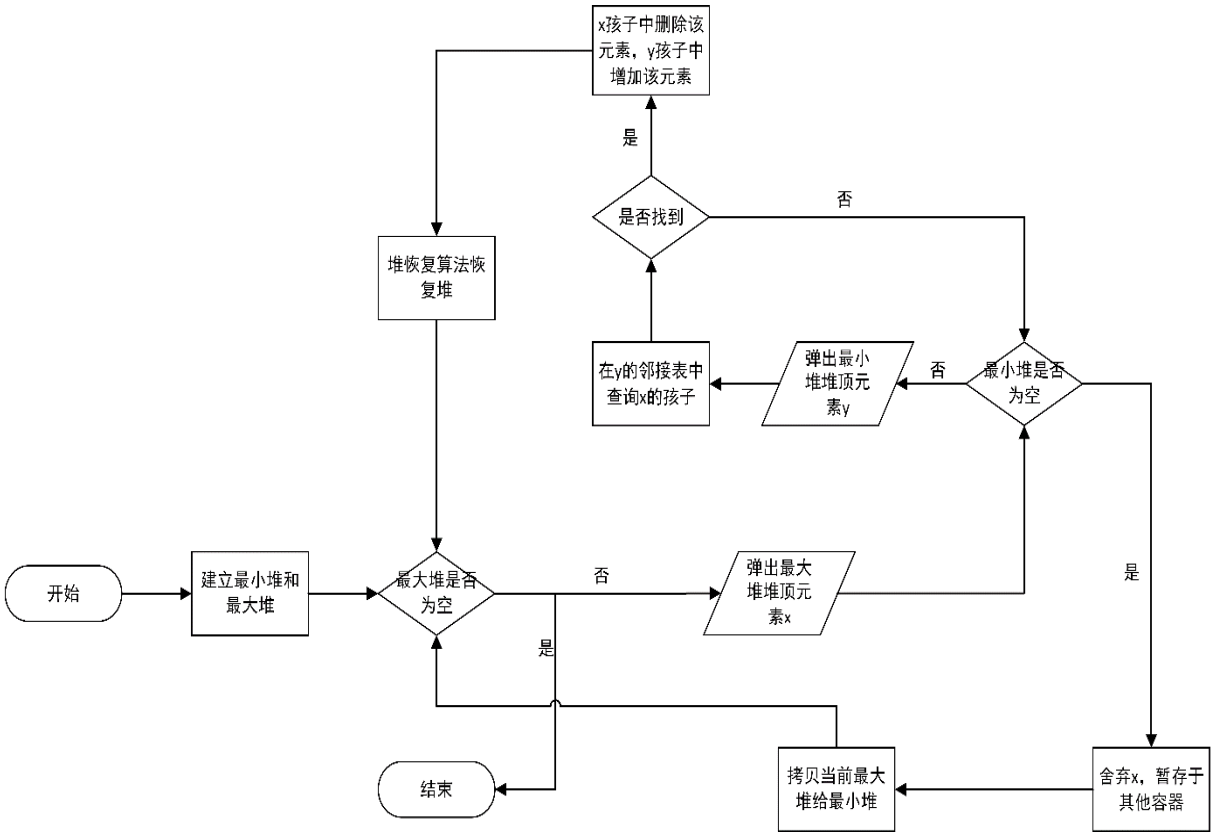

图 9 最大堆和最小堆联合维护算法流程图

5.2.5 网络信号发送方案规划

对于信息发送方案，我们假设第一次发送信息时的时刻为 0。

根据我们以上建立的多叉树模型父子节点关系，我们可以新建一个邻接表，并用该邻接表通过程序(9.1.1 问题二代码三)建立完整的最终树形模型。在该树形结构中，利用 BFS 算法从基站出发，每次可以访问数量为第一层节点数量的子节点，从而得到到达每一个节点所需要的访问次数，访问次数减去该节点所在树的高度，即该节点发送信息的时间。

最终可得到各节点的发送时间，如下表所示：

表 11 发送节点时刻表

时刻	发送节点	时刻	发送节点	时刻	发送节点
0	42 43 47 48 67 68 72 73 74 22 37 18 38 49 17 41 46 23	1	98 44 62 97 63 66 71 92 69 93 45	2	50 70 75 94 95 100 99 117 118 119 16 32 13 33 39 20 12
3	122 123 124 36 21 19 57 96 121 58 61 91 112	4	64 88 40 24 25 65 90 114 115 125 11 27 8 28 35	5	120 87 113 89 116 15 7 31 14 52 82 53 56 111 107
6	59 83 34 60 84 109 85 108 110 86 6 26 3 29 30 10 2 9 77 54	7	78 51 106 102 103 55 105 80 104 79 4 5 1 101 76	8	81

5.2.6 程序流程图

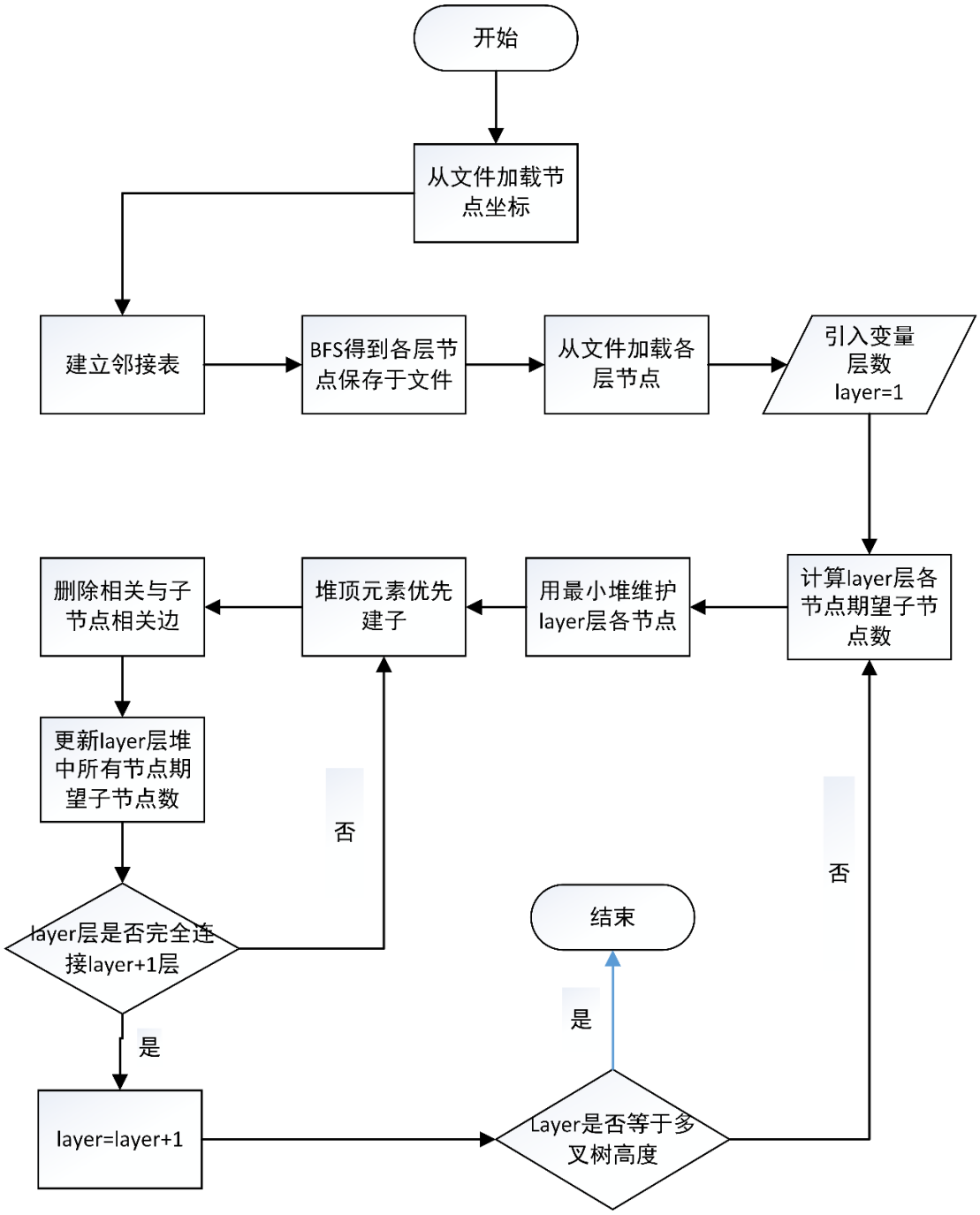

图 10 问题二程序流程图

5.3 问题三

节点信号发送时的信息的平均时间为$T_{send}=0.1s$，接收数据的平均时间为$T_{acc}=0.1s$，信息转发的所需时间$T_{dis}=4.5\times10^{-5}s$，节点发送数据的能耗$P_{send}=3\times10^{-2}W$，节点接收数据的能耗$P_{acc}=2\times10^{-2}W$，对于第一层的节点，其每一个发送周期（所有子节点和自身都将数据传给基站 B.S）所需要消耗的能量为：

$$E_{\Re}=T_{send}P_{send}+T_{acc}P_{acc}+COUNT(son[\Re])(T_{dis}P_{send}+T_{acc}P_{acc}) \quad (12)$$

其周期为：

$$T_{all} = T_{send} + T_{acc} + COUNT(son[\Re])(T_{dis} + T_{acc}) \quad (13)$$

那么对于该节点的平均功率为：

$$P_{ave} = \frac{E_{\Re}}{T_{all}} \quad (14)$$

P_{ave} 随着 $COUNT(son[\Re])$ 的增加而增加，所以子节点数越多，该点平均功率越大。由此我们可以找到问题 2 中子树最大的第一层节点，即最早失效点。

利用程序(9.1.1 问题三代码一)计算得到各节点子树大小和可通信时间如下：

表 12 节点子树与可通信时间关系表

子树根	42	43	47	48	67	68	72	73	74	98
子树大小	9	18	11	6	10	6	14	8	25	18

可以计算得到各节点可通信时间为：

表 13 各节点子树根可通信时间表

子树根	可通信时间（s）
42	1429.77
43	1458.26
47	1439.39
48	1406.86
67	1434.96
68	1406.86
72	1449.44
73	1423.59
74	1467.53
98	1458.26

分析该表我们可以选择在 48 和 68 节点所在位置各增加一个节点。

六．模型灵敏度分析

6.1 模型鲁棒性分析

对于问题一，我们使用网格划分的方式将传感器节点对海域的覆盖率转化为对基点的覆盖率问题。

该转化使得求解覆盖海域的传感器数量问题可推广到任何一个比率。另外，通过震荡初始点位置的方式，也使得最终的结果区间能够在一定程度上趋近最优解， BFS 的使用保障了整个网络存在生成树并且这个网络一定可通信。相对而言，问题一算法的鲁棒性尚可。

问题二中我们引入期望子节点的概念并采用最小堆维护得到初始树形模型，再使用最小堆和最大堆联合维护的方式消除期望子节点存在的问题以保障树形结构的最优，该算法在时间复杂度和空间复杂度上具有较高的效率，而且多重保障机制使得结果一定趋

于最优。所以，问题二算法的鲁棒性较高。

6.2 模型灵敏度分析

问题一中，算法每次都会剔除无用点，这使算法在求解覆盖率较低的情况下拥有更高的效率。我们的算法在求解 100%全覆盖的时候，程序对于结果的求解较慢，平均运行 3000 次可以得到一个有效解，求解 90%覆盖率的问题时，程序在 10 分钟后可得稳定的最终解为 79，求解 80%覆盖率的问题时，我们的代码在 1 分钟后可得稳定的最优解 64。

问题二中，在利用 BFS 求解各层节点的时候，因为使用了染色策略，使得每个点都只能被访问一次，那么此时需要访问的次数为 $\sum_{i=0}^{\Omega} size(adj\_list[i])$ 。同时引入期望子节点数的概念，使得可以利用最小堆维护树的建立，假设最小堆大小为 $\text{size}(\min\_pile)$，那么建树的时间度为 $\Omega\text{size}(\min\_pile)\log[size(\min\_pile)]$，建树过程也有较高的效率。最后在利用最大最小堆联合维护微调树链的时候，我们的时间复杂度为 $\text{size}(\min\_pile)\text{size}(\max\_pile)$，整体来看，算法具有较高的效率和较低的时间复杂度。

七．模型的评价及推广

7.1 模型的优点

1）**考虑因素多，鲁棒性强。**

本模型采用多叉树的结构管理所有的水下传感器节点，分层式的管理使得整个网络的通信效率较高，堆维护建树的过程也使最终的树结构中各节点的压力区域平均，系统可以运行更长的时间，也更为节能。

2）**多种算法组合使用，可靠性强**

本模型使用了多种图论树论算法和各类数据结构保障模型的可靠性，并采取可多重维护机制使得模型趋于最优。

引入了自定义的变量期望子节点数，并使用自创的最小最大堆联合维护算法保障可靠性。

7.2 模型的不足

本模型在求解满足一定覆盖率下的最少节点这个问题时，使用网格遍历的方式遍历网格交点，实际解会存在误差。另外，我们采取的区域内随机的方式，导致结果最终只能指向局部最优解，且随着覆盖率的增加，本算法所得解会更偏离实际解。这是本模型的最大不足，也是待提升之处。

7.3 模型的推广

1） 本模型所采用的平均子节点的多叉树建立方式和最小最大堆联合维护算法可以作为数据结构树结构一种参考方式。

2） 本模型所采用的固定窗口的 BFS 算法可以使用在路径探测，智能寻路等领域。

八．参考文献

[1] 王一楠.水下无线传感器的节点部署策略和算法[D]. 南京邮电大学，2013.

[2] 刘俊.水下无线传感网络节点部署算法研究[D]. 杭州电子科技大学，2016.

[3] Bálint, Vojtech. On the maximum number of points at least one unit away from each other in the unit n-cube[]. Periodica Mathematica Hungarica，2008:164-170.

[4] Circle Covering Squares.[DB/OL]. https://www2.stetson.edu/~efriedma/circovsqu/#opennewwindow.

[5] 周凯，邬学军，宋军全.数学建模[M]. 浙江大学出版社，2017：21-65.

[6] 姜启源，谢金星，叶俊. 数学模型（第三版）[M]. 北京：高等教育出版社，2003：274-324.

[7] 刘汝佳.算法竞赛入门经典（第二版）[M]. 清华大学出版社，2014:355-358，392，164-166.

[8] Thomas H. Cormen，Charles E. Leiserson，Ronald L. Rivest，Clifford Stein.算法导论（第三版）[M]. 机械工业出版社，2013：90-92.

[9] 秋叶拓哉，岩田阳一，北川宜稔.挑战程序设计竞赛（第二版）[M]. 人民邮电出版社，2013.

九. 附录

9.1 代码

9.1.1 C++代码

问题一

代码一：

```
#ifndef MCM_VECTOR3_H
#define MCM_VECTOR3_H

class Vector3
{
public:
    float x, y, z;
    Vector3(float x, float y, float z)
            :x(x), y(y), z(z){}
    Vector3() = default;

    bool operator==(const Vector3& another)
    {
        if(x == another.x && y == another.y && z == another.z)
            return true;
        return false;
    }
};

#endif //MCM_VECTOR3_H
```

```
#include <iostream>
#include "Vector3.h"
#include <vector>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <queue>
#include <fstream>
#include <set>
#include <climits>
```

```
using namespace std;

#define PERCENT 0.2
#define RADIUS 20
#define MAX_N 1030301
float height = 40/sqrt(3);
Vector3 base(0, 0, 0);
vector<Vector3> basePos;
vector<Vector3> basePosCopy;
bool vis[MAX_N];
int c[MAX_N + 1];

void Pre()
{
    ofstream file("/home/cloud/document/ques2_data.txt");
    file.precision(3);
    memset(vis, 0, sizeof(vis));
    memset(c, 0, sizeof(c));
    Vector3 temp;
    for(float i = -50 + height/2; i - height / 2 < 50;i += height)
        for(float j = -100 + height/2; j - height < 0; j += height)
            for(float k = -50 + height / 2; k - height / 2 < 50 ; k +=
height)
            {
                float ii = i, jj = j, kk = k;
                //if(ii > 50) ii = 50 - height/2;
                //if(jj > 0) jj = -height/2;
                //if(kk > 50) kk = 50 - height/2;
                basePos.push_back(Vector3(ii, jj, kk));
                basePosCopy.push_back(Vector3(ii, jj, kk));
                file << fixed << ii << ' ' << fixed << jj << ' ' << fixed
<< kk << endl;
                printf("%f %f %f\n", i, j, k);
            }
    cout << "需要点的个数为：" << basePos.size() << endl;
}

///
/// 判断是否在半径之内
/// 全覆盖判断
/// \param _check
/// \param center
/// \return
```

```
bool check(const Vector3& _check, const Vector3& center)
{
    if(pow(_check.x - center.x, 2) + pow(_check.y - center.y, 2)
       +pow(_check.z - center.z, 2) > pow(RADIUS, 2))
        return false;
    return true;
}


bool contained(const Vector3& _check)
{
    for(int i = 0; i < basePos.size(); ++i)
    {
        if(pow(_check.x - basePos[i].x, 2) + pow(_check.y - basePos[i].y,
2)
           + pow(_check.z - basePos[i].z, 2) <= 400)
            return true;
    }
    return false;
}

int vis_all()
{
    int count = 0;
    for(int i = -50; i <= 50; ++i)
    {
        for(int j = -100; j <= 0; ++j)
            for(int k = -50; k <= 50; ++k)
                if(!contained(Vector3(i, j, k)))
                    count++;
    }
    if(count <= MAX_N * PERCENT)
        return count;
    return 0;
}

void shake()
{
    //srand(TIME_UTC);
    for(auto it = basePos.begin(); it != basePos.end(); ++it)
    {
        int delta_x = ((rand()%4000) - 2000) / 100.0;
        int delta_y = ((rand()%4000) - 2000) / 100.0;
        int delta_z = ((rand()%4000) - 2000) / 100.0;
```

```
        (*it).x += delta_x;
        (*it).y += delta_y;
        (*it).z += delta_z;
    }
}

void Resume()
{
    basePos = basePosCopy;
    memset(vis, 0, sizeof(vis));
}

float distance(const Vector3& a, const Vector3& b)
{
    return sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2) + pow(a.z - b.z,
2));
}



bool out_bound(const Vector3& _check)
{

    if(_check.x - RADIUS > 50 || _check.y - RADIUS > 0 || _check.z -
RADIUS > 50
       || _check.x + RADIUS < -50 || _check.y + RADIUS < -100 || _check.z
+ RADIUS < -50)
        return true;
    return false;
}



typedef long long LL;
LL fail_time = 0;

bool color_mesh[130];
vector<int> next_list[130];



bool bfs()
{
    queue<int> que;
    que.push(0);
    color_mesh[0] = 1;
    while(!que.empty())
```

```
    {
        int t = que.front();
        que.pop();
        color_mesh[t] = 1;
        for(int i = 0; i < next_list[t].size(); ++i)
        {
            if(!color_mesh[next_list[t][i]])
                que.push(next_list[t][i]);
        }
    }
    for(int i = 0; i <= basePos.size(); ++i)
    {
        if(!color_mesh[i])
            return false;
    }
    return true;
}

bool connect()
{
    memset(color_mesh, 0, sizeof(color_mesh));
    vector<Vector3> eraseList;
    vector<Vector3> eraseListCopy;
    for(int i = 0; i < basePos.size(); ++i)
    {
        if(out_bound(basePos[i]))
        {
            eraseList.push_back(basePos[i]);
            eraseListCopy.push_back(basePosCopy[i]);
        }
    }
    for(int i = 0; i < eraseList.size(); i++)
    {
        for (auto it = basePos.begin(); it != basePos.end(); ++it)
        {
            if (eraseList[i] == *it)
            {
                basePos.erase(it);
                break;
            }
        }
    }
    //printf("当前 basePos 大小为%d\n", basePos.size());
    for(int i = 0; i < basePos.size(); ++i)
```

```
    {
        for(int j = i + 1; j < basePos.size(); ++j)
        {
            if(distance(basePos[i], basePos[j]) <= 30)
            {
                next_list[i + 1].push_back(j + 1);
                next_list[j + 1].push_back(i + 1);
            }
        }
    }
    for(int i = 0; i < basePos.size(); ++i)
    {
        if(distance(basePos[i], base) <= 35)
        {
            next_list[0].push_back(i + 1);
            next_list[i + 1].push_back(0);
        }
    }
    //printf("邻接表建立完成\n");
    int return_val = bfs();
    if(return_val)
    {
        for(int i = 0; i < eraseListCopy.size(); ++i)
        {
            for(auto it = basePosCopy.begin(); it != basePos.end(); ++it)
                if(eraseListCopy[i] == *it)
                {
                    basePosCopy.erase(it);
                    break;
                }
        }
    }
    return return_val;
}


///ques1
int main()
{
    ofstream fileout("/home/cloud/document/data80.txt", ios::app);
    Pre(); int min_ans = INT_MAX;
    while(true)
    {
        shake(); int miss_vertex = vis_all();
```

```
        if(miss_vertex && connect())
        {
            min_ans = min(min_ans, (int)basePosCopy.size());
            printf("现在只需要%d 个点  当前最少为%d\n",
basePosCopy.size(), min_ans);
            if(basePosCopy.size() <= 125)
            {
                fileout << "当前所需节点数量为: " << basePosCopy.size()
<< endl;
                fileout << "缺失节点数为" << miss_vertex << endl;
                fileout << "各点坐标如下:\n";
                for(int i = 0; i < basePos.size(); ++i)
                {
                    fileout << basePos[i].x << "   " << basePos[i].y << "
" << basePos[i].z << endl;
                }
                fileout << "\n\n\n";
                cout << "已将该答案输出给文件\n\n";
            }
        }
        else
        {
            fail_time++;
           // if(fail_time%100000==0)
                cout <<"失败" <<fail_time <<"次\n";
        }
        Resume();
    }
}
```

问题二

代码一：

```
#include <iostream>
#include "Vector3.h"
#include <vector>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <queue>
#include <fstream>
#include <set>
#include <climits>

using namespace std;
```

```
#define PERCENT 0.2
#define RADIUS 20
#define MAX_N 1030301
float height = 40/sqrt(3);
Vector3 base(0, 0, 0);
vector<Vector3> basePos;
vector<Vector3> basePosCopy;
bool vis[MAX_N];
int c[MAX_N + 1];

///
/// 判断是否在半径之内
/// 全覆盖判断
/// \param _check
/// \param center
/// \return
bool check(const Vector3& _check, const Vector3& center)
{
    if(pow(_check.x - center.x, 2) + pow(_check.y - center.y, 2)
       +pow(_check.z - center.z, 2) > pow(RADIUS, 2))
        return false;
    return true;
}


bool contained(const Vector3& _check)
{
    for(int i = 0; i < basePos.size(); ++i)
    {
        if(pow(_check.x - basePos[i].x, 2) + pow(_check.y - basePos[i].y,
2)
           + pow(_check.z - basePos[i].z, 2) <= 400)
            return true;
    }
    return false;
}

int vis_all()
{
    int count = 0;
    for(int i = -50; i <= 50; ++i)
    {
        for(int j = -100; j <= 0; ++j)
```

```
            for(int k = -50; k <= 50; ++k)
                if(!contained(Vector3(i, j, k)))
                    count++;
    }
    if(count <= MAX_N * PERCENT)
        return count;
    return 0;
}

void Resume()
{
    basePos = basePosCopy;
    memset(vis, 0, sizeof(vis));
}

float distance(const Vector3& a, const Vector3& b)
{
    return sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2) + pow(a.z - b.z,
2));
}


bool out_bound(const Vector3& _check)
{

    if(_check.x - RADIUS > 50 || _check.y - RADIUS > 0 || _check.z -
RADIUS > 50
       || _check.x + RADIUS < -50 || _check.y + RADIUS < -100 || _check.z
+ RADIUS < -50)
        return true;
    return false;
}


typedef long long LL;
LL fail_time = 0;

bool color_mesh[130];
vector<int> next_list[130];

/*
 *
 * QUES2
*/
```

```
vector<int> child[130];//每个节点其所下联网络
set<int> nodeTable[130];//邻接表
vector<Vector3> nodePos;     //每个节点的坐标
bool isUsed[130];
vector<int> nodePerCeng[10];

/*优先考虑邻接表较小的节点，其优先选择下属孩子*/

/// 计算点 x 的期望子节点数量
/// \param x
/// \return
float calculate_except(int x, int ceng)
{
    //cout << "开始计算期望\n";
    float ans = 0;
    ans+=child[x].size();
    //计算同层还有多少节点可以访问 x 可以访问的节点。
    //cout << "x 可访问点有" << nodeTable[x].size() << "个\n";
    //printf("x 所在层节点数为%d 个\n", nodePerCeng[ceng].size());
    for(auto it = nodeTable[x].begin(); it != nodeTable[x].end(); ++it)
//x 可访问点
    {
        float count = 0;
        for(int j = 0; j < nodePerCeng[ceng].size(); ++j) //x 所在层全部点
        {
            if(nodePerCeng[ceng][j] == x) continue;
            for(auto k = nodeTable[nodePerCeng[ceng][j]].begin();
                    k != nodeTable[nodePerCeng[ceng][j]].end(); ++k) //x
所在层某一点可访问点
            {

                //printf("x 所在层某一点可访问点%d\n",
nodeTable[nodePerCeng[ceng][j]].size());
                for(int ttt = 0; ttt < nodePerCeng[ceng+1].size(); ++ttt)
//x 下一层点
                {
                    if(*it == nodePerCeng[ceng+1][ttt]
                            &&*k == nodePerCeng[ceng+1][ttt])
                        count++;
                }
            }
        }
```

```
        if(count)
            ans += 1/count;
    }
    //cout << "计算期望成功\n";
    return ans;
}

struct node
{
    int rank;
    int value;
};

void BFS_Search()
{
    ofstream file("/home/cloud/document/层次.txt");
    //所有标记点为不可达点
    queue<node> que;
    node temp;
    temp.value = 0;
    temp.rank = 0;
    isUsed[0] = 1;
    que.push(temp); int laz = 0;
    while(!que.empty())
    {
        node start = que.front();
        que.pop();

        if(start.rank == laz)
        {
            if(!isUsed[start.value])
                file << start.value << ' ';
        }
        else
        {
            file << "第" << laz + 1 << "层:\n";
            if(!isUsed[start.value])
                file << start.value << ' ';
            laz = start.rank;
        }
        isUsed[start.value] = 1;

        for(auto it = nodeTable[start.value].begin(); it !=
nodeTable[start.value].end(); ++it)
```

```
        {
            if(!isUsed[*it])
            {
                node q;
                q.value = *it;
                q.rank = start.rank + 1;
                que.push(q);
            }
        }
    }
}

void Read_File()
{
    int laz = 1;
    int x;
    ifstream filein("/home/cloud/document/层次.txt");
    while(filein >> x)
    {
        if(x == -2) break;
        if(x == -1)
        {
            laz++;
            continue;
        }
        nodePerCeng[laz].push_back(x);
    }
    filein.close();
}


struct exp_node
{
    float exp;
    int value;
};

struct cmp
{
    bool operator()(const exp_node& a, const exp_node& b)
    {
        return a.exp > b.exp;
    }
};
```

```
void build_connect_table()
{
    memset(isUsed, 0, sizeof(isUsed));
    //------------------------------------------------------------从文件
读取节点坐标
    ifstream filein("/home/cloud/document/ques2_data.txt");
    float x, y, z;
    while(filein >> x >> y >> z)
    {
        if(x >= 50) x=50-height/2;
        if(y >= 0) y = 0-height/2;
        if(z >= 50) z = 50 - height/2;
        Vector3 temp_pos(x,y,z);
        nodePos.push_back(temp_pos);
    }
    filein.close();
    //------------------------------------------------------------坐标读
取结束

    //------------------------------------------------------------建立邻
接表
    for(int i = 0; i < nodePos.size(); ++i)
    {
        for(int j = i + 1; j < nodePos.size(); ++j)
        {
            if(distance(nodePos[i], nodePos[j]) <= 30)
            {
                nodeTable[i+1].insert(j + 1);
                nodeTable[j+1].insert(i + 1);//建立邻接表
            }
        }
    }
    for(int i = 0; i < nodePos.size(); ++i)
    {
        if(distance(nodePos[i], base) <= 35)
        {
            nodeTable[0].insert(i + 1);
            nodeTable[i + 1].insert(0);
        }
    }
    cout << "邻接表建立完全\n";
    //------------------------------------------------------------------以
上邻接表建立完全
```

```
    //根据邻接表用BFS进行遍历 获取各层点的坐标。
    //BFS_Search();
    Read_File();
    cout << "文件读取成功\n";




////////////////////////////////////////////////////////////////////////////
////单元测试

////////////////////////////////////////////////////////////////////////////
////地一层每一个元素最多儿子数量
    auto f = [&]()
    {
        for (int i = 0; i < nodePerCeng[1].size(); i++)
        {
            vector<int> son_vector;
            int temp_node = nodePerCeng[1][i];
            for (auto it = nodeTable[temp_node].begin(); it !=
nodeTable[temp_node].end(); ++it)
            {
                for (int j = 0; j < nodePerCeng[2].size(); ++j)
                {
                    if (*it == nodePerCeng[2][j])
                    {
                        son_vector.push_back(*it);
                    }
                }
            }

            cout << temp_node << "的子节点有: ";
            for (int ii = 0; ii < son_vector.size(); ++ii)
            {
                cout << son_vector[ii] << " ";
            }
            cout << endl;
            son_vector.clear();
        }
        cout << endl << endl;
    };
    f();
```

```
////////////////////////////////////////////////////////////////////////////
////

////////////////////////////////////////////////////////////////////////////
////




    for(int cengshu = 1; cengshu < 6; cengshu++) {

        priority_queue<exp_node, vector<exp_node>, cmp> pri_que;
        for (int i = 0; i < nodePerCeng[cengshu].size(); ++i) {
            int curr_node = nodePerCeng[cengshu][i];
            exp_node temp;
            temp.exp = calculate_except(curr_node, cengshu);
            temp.value = curr_node;
            pri_que.push(temp);
        }
        int num = 0;
        while (num < nodePerCeng[cengshu + 1].size()) {

////////////////////////////////////////////////////////////////////////////
//单元测试

////////////////////////////////////////////////////////////////////////////
//
            auto f2 = [&]() {
                vector<exp_node> test_vector;
                while (!pri_que.empty()) {
                    exp_node temp_node = pri_que.top();
                    pri_que.pop();
                    test_vector.push_back(temp_node);
                    cout << temp_node.value << "的期望是: " <<
temp_node.exp << endl;
                }
                for (int i = 0; i < test_vector.size(); i++) {
                    pri_que.push(test_vector[i]);
                }
            };
            //f2();
```

```
////////////////////////////////////////////////////////////////////////////
//

////////////////////////////////////////////////////////////////////////////
//
            exp_node t = pri_que.top();
            pri_que.pop();
            //找 t 所能去的下一层的任何一个节点
            int flag = -1;
            auto it = nodeTable[t.value].end();
            for (int i = 0; i < nodePerCeng[cengshu + 1].size(); ++i) {
                for (auto j = nodeTable[t.value].begin(); j !=
nodeTable[t.value].end(); ++j) {
                    if (nodePerCeng[cengshu + 1][i] == *j)  //child
                    {
                        flag = *j;
                        child[t.value].push_back(*j);
                        it = j;
                        break;
                    }
                }
                if (flag != -1) break;
            }
            int des = flag;
            if (it != nodeTable[t.value].end())
                nodeTable[t.value].erase(it);
            else {
                continue;
            }
            //删除 des 点的链接边
            for (int i = 0; i < nodePerCeng[cengshu].size(); ++i) {
                for (auto ite =
nodeTable[nodePerCeng[cengshu][i]].begin();
                     ite != nodeTable[nodePerCeng[cengshu][i]].end();
++ite) {
                    if (*ite == des) {
                        nodeTable[nodePerCeng[cengshu][i]].erase(ite);
                        break;
                    }
                }
            }//开始更新 t 点的 exp
            t.exp = calculate_except(t.value, cengshu);
            vector<exp_node> temp_vector;
```

```
            temp_vector.push_back(t);
            while (!pri_que.empty()) {
                exp_node tx_node = pri_que.top();
                pri_que.pop();
                tx_node.exp = calculate_except(tx_node.value, cengshu);
                temp_vector.push_back(tx_node);
            }
            for (int i = 0; i < temp_vector.size(); ++i) {
                pri_que.push(temp_vector[i]);
            }
            num++;


////////////////////////////////////////////////////////////////////////////
单元测试

////////////////////////////////////////////////////////////////////////////
输出各节点儿子数量
            auto f3 = [&]() {
                for (int i = 0; i < nodePerCeng[cengshu].size(); i++) {
                    cout << "点" << nodePerCeng[cengshu][i] << "的儿子有:
";
                    for (int j = 0; j <
child[nodePerCeng[cengshu][i]].size(); ++j)
                        cout << child[nodePerCeng[cengshu][i]][j] << ' ';
                    cout << endl;
                }
                cout << endl;
                cout << endl;
            };
            if (num == nodePerCeng[cengshu + 1].size())
                f3();

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
        }
    }
}
int main()
{
    build_connect_table();
}
```

该代码所需文件数据如下：

层次.txt

```
42 43 47 48 67 68 72 73 74 98 -1
17 22 37 41 46 18 23 38 44 49 62 66 71 92 97 63 69 93 45 50 70 75 94 95
99 100 117 118 119 122 123 124 -1
12 16 21 32 36 13 19 24 33 39 20 25 40 57 61 87 91 96 112 116 121 58 64
88 65 113 89 90 114 115 120 125 -1
7 11 27 31 8 14 15 28 34 35 52 56 82 86 107 111 53 59 83 60 108 84 85 109
110 -1
2 6 26 3 9 10 29 30 51 77 81 102 106 54 78 55 103 79 80 104 105 -1
1 4 5 76 101 -2
```

ques2_data.txt

```
-38.453 -88.453 -38.453
-38.453 -88.453 -15.359
-38.453 -88.453 7.735
-38.453 -88.453 30.829
-38.453 -88.453 53.923
-38.453 -65.359 -38.453
-38.453 -65.359 -15.359
-38.453 -65.359 7.735
-38.453 -65.359 30.829
-38.453 -65.359 53.923
-38.453 -42.265 -38.453
-38.453 -42.265 -15.359
-38.453 -42.265 7.735
-38.453 -42.265 30.829
-38.453 -42.265 53.923
-38.453 -19.171 -38.453
-38.453 -19.171 -15.359
-38.453 -19.171 7.735
-38.453 -19.171 30.829
-38.453 -19.171 53.923
-38.453 3.923 -38.453
-38.453 3.923 -15.359
-38.453 3.923 7.735
-38.453 3.923 30.829
-38.453 3.923 53.923
-15.359 -88.453 -38.453
-15.359 -88.453 -15.359
-15.359 -88.453 7.735
-15.359 -88.453 30.829
-15.359 -88.453 53.923
-15.359 -65.359 -38.453
```

-15.359 -65.359 -15.359
-15.359 -65.359 7.735
-15.359 -65.359 30.829
-15.359 -65.359 53.923
-15.359 -42.265 -38.453
-15.359 -42.265 -15.359
-15.359 -42.265 7.735
-15.359 -42.265 30.829
-15.359 -42.265 53.923
-15.359 -19.171 -38.453
-15.359 -19.171 -15.359
-15.359 -19.171 7.735
-15.359 -19.171 30.829
-15.359 -19.171 53.923
-15.359 3.923 -38.453
-15.359 3.923 -15.359
-15.359 3.923 7.735
-15.359 3.923 30.829
-15.359 3.923 53.923
7.735 -88.453 -38.453
7.735 -88.453 -15.359
7.735 -88.453 7.735
7.735 -88.453 30.829
7.735 -88.453 53.923
7.735 -65.359 -38.453
7.735 -65.359 -15.359
7.735 -65.359 7.735
7.735 -65.359 30.829
7.735 -65.359 53.923
7.735 -42.265 -38.453
7.735 -42.265 -15.359
7.735 -42.265 7.735
7.735 -42.265 30.829
7.735 -42.265 53.923
7.735 -19.171 -38.453
7.735 -19.171 -15.359
7.735 -19.171 7.735
7.735 -19.171 30.829
7.735 -19.171 53.923
7.735 3.923 -38.453
7.735 3.923 -15.359
7.735 3.923 7.735
7.735 3.923 30.829
7.735 3.923 53.923

30.829 -88.453 -38.453
30.829 -88.453 -15.359
30.829 -88.453 7.735
30.829 -88.453 30.829
30.829 -88.453 53.923
30.829 -65.359 -38.453
30.829 -65.359 -15.359
30.829 -65.359 7.735
30.829 -65.359 30.829
30.829 -65.359 53.923
30.829 -42.265 -38.453
30.829 -42.265 -15.359
30.829 -42.265 7.735
30.829 -42.265 30.829
30.829 -42.265 53.923
30.829 -19.171 -38.453
30.829 -19.171 -15.359
30.829 -19.171 7.735
30.829 -19.171 30.829
30.829 -19.171 53.923
30.829 3.923 -38.453
30.829 3.923 -15.359
30.829 3.923 7.735
30.829 3.923 30.829
30.829 3.923 53.923
53.923 -88.453 -38.453
53.923 -88.453 -15.359
53.923 -88.453 7.735
53.923 -88.453 30.829
53.923 -88.453 53.923
53.923 -65.359 -38.453
53.923 -65.359 -15.359
53.923 -65.359 7.735
53.923 -65.359 30.829
53.923 -65.359 53.923
53.923 -42.265 -38.453
53.923 -42.265 -15.359
53.923 -42.265 7.735
53.923 -42.265 30.829
53.923 -42.265 53.923
53.923 -19.171 -38.453
53.923 -19.171 -15.359
53.923 -19.171 7.735
53.923 -19.171 30.829

```
53.923 -19.171 53.923
53.923 3.923 -38.453
53.923 3.923 -15.359
53.923 3.923 7.735
53.923 3.923 30.829
53.923 3.923 53.923
```

==========分割线==========

代码二：

基于最小堆和最大堆的联合维护优化代码如下所示

```
#include <fstream>
#include <iostream>
#include <vector>
#include <queue>
#include <set>
using namespace std;


struct node
{
    int value;
    set<int> son;
};

struct big_cmp
{
    bool operator()(const node& a, const node& b)
    {
        return a.son.size() < b.son.size();
    }
};

struct small_cmp
{
    bool operator()(const node& a, const node& b)
    {
        return a.son.size() > b.son.size();
    }
};

set<int> adj_list[130];
vector<node> true_list;

void read_data()
```

```
{
    ifstream file("data.txt");
    int x; bool flag = 0;
    int y;
    int t;
    for (int i = 0; i < 10; ++i)
    {
        file >> x >> t;
        for (int j = 0; j < t; ++j)
        {
            file >> y;
            adj_list[x].insert(y);
        }
    }
    file.close();
}

void read_true()
{
    ifstream file("true.txt");
    int x; bool flag = 0;
    int y;
    int t;
    for (int i = 0; i < 10; ++i)
    {
        file >> x >> t;
        node _node;
        _node.value = x;
        for (int j = 0; j < t; ++j)
        {
            file >> y;
            _node.son.insert(y);
        }
        true_list.push_back(_node);
    }
    file.close();
}

void test_print(priority_queue<node, vector<node>, big_cmp>& que)
{
    vector<node> saveVector;
    while (!que.empty())
    {
        saveVector.push_back(que.top());
```

```
        printf("%d的子节点有: ",que.top().value);
        for (auto it = que.top().son.begin(); it != que.top().son.end();
++it)
        {
            cout << ' ' << *it;
        }
        cout << endl;
        que.pop();
    }
    for (int i = 0; i < saveVector.size(); ++i)
    {
        que.push(saveVector[i]);
    }
}

#define QUEUE_SIZE 10

void deal()
{
    priority_queue<node, vector<node>, big_cmp> que;
    priority_queue<node, vector<node>, big_cmp> que1;
    priority_queue<node, vector<node>, small_cmp> que2;
    for (int i = 0; i < true_list.size(); ++i)
    {
        que1.push(true_list[i]);
        que2.push(true_list[i]);
    }
    bool flag = 0; int begin_size = QUEUE_SIZE;
    while (!que1.empty())
    {
        flag = 0;
        node t_node = que1.top();
        que1.pop(); int count = 0;
        //-------------------------------------------------最多子节点孩
子的遍历
        for (auto i = t_node.son.begin(); i != t_node.son.end(); ++i)
        {
            int des = *i; //des是下面的儿子
            //-----------------------------------------对于点index遍
历
            while (!que2.empty())
            {
                node _node = que2.top();
                que2.pop();
```

```
                for (auto it = adj_list[_node.value].begin();
                    it != adj_list[_node.value].end(); ++it)
                {
                    //*it是上面的儿子
                    if (*it == des && t_node.son.size() -
_node.son.size() >= 2)
                    {
                        cout << "hahahahahaahahah\n";
                        //下面删除儿子
                        t_node.son.erase(i);
                        //另一个增加儿子
                        _node.son.insert(des);

                        //-------------------------------------------------
-------
                        vector<node> storage;
                        while (!que2.empty())
                        {
                            storage.push_back(que2.top());
                            que2.pop();
                        }
                        for (int xixi = 0; xixi < storage.size() - 1;
xixi++)
                        {
                            que2.push(storage[xixi]);
                        }
                        //-----------------------------------------------删
除que2尾部元素

                        que2.push(_node);
                        que2.push(t_node);
                        count = que2.size();
                        int need_count = QUEUE_SIZE - count;
                        while (que1.size() > need_count) que1.pop();
                        while (!que1.empty())
                        {
                            node temp = que1.top();
                            que1.pop();
                            que2.push(temp);
                        }

                        //将que2赋值给que1
                        vector<node> temp_vector;
                        while (!que2.empty())
```

```
                    {
                        temp_vector.push_back(que2.top());
                        que2.pop();
                    }
                    for (int iii = 0; iii < temp_vector.size(); ++iii)
                    {
                        que1.push(temp_vector[iii]);
                        que2.push(temp_vector[iii]);
                    }
                    //que1和que2已经恢复。
                    flag = 1;
                    break;
                }
            }
            if (flag) break;
        }

        while (!que2.empty()) que2.pop();
        //-------------------------------------------------------------
----
        //-------------------------------------------------------------
-----
        vector<node> temp_vector;
        while (!que1.empty())
        {
            temp_vector.push_back(que1.top());
            que1.pop();
        }
        for (int iii = 0; iii < temp_vector.size(); ++iii)
        {
            que1.push(temp_vector[iii]);
            que2.push(temp_vector[iii]);
        }
        //-------------------------------------------------------------
----
        //-------------------------------------------------------------
----
        if (flag) break;
    }
    if (begin_size != que1.size())
    {
        que.push(t_node);
    }
    cout << "que1大小为" << que1.size() << endl;
```

```
        cout << "que2大小为" << que2.size() << endl;
        test_print(que);
        cout << endl;
        test_print(que1);
        cout << endl;
        cout << endl;
        cout << "-----------------------------------------------------------
----------\n";
    }
}

int main()
{
    read_data();
    read_true();
    deal();
    system("pause");
}
```

此代码所需文件数据如下：

data.txt

```
42 5 17 22 37 41 46
43 5 18 23 38 44 49
47 4 17 22 41 46
48 4 18 23 44 49
67 5 62 66 71 92 97
68 3 63 69 93
72 4 66 71 92 97
73 2 69 93
74 11 44 45 49 50 69 70 75 94 95 99 100
98 11 92 93 94 97 99 117 118 119 122 123 124
```

true.txt

```
42 2 22 37
43 3 18 38 49
47 3 17 41 46
48 2 23 44
67 2 62 97
68 1 63
72 2 66 71
73 2 69 93
74 6 45 50 70 75 95 100
98 9 92 94 99 117 118 119 122 123 124
```

==========BFS 查询发送时间==========

代码三：

```
#include <iostream>
#include <fstream>
#include <vector>
#include <queue>
using namespace std;

class node
{
public:
    int rank, value;
    node() = default;
    node(int _rank, int _value) :rank(_rank), value(_value) {}
};

vector<int> adj_list[130];

void read_file()
{
    ifstream file("adj_list.txt");
    int x, t, y;
    while (file >> x)
    {
        file >> t;
        for (int i = 0; i < t; ++i)
        {
            file >> y;
            adj_list[x].push_back(y);
        }
    }
}

vector<int> ans[10];

void bfs()
{
    queue<node> que; int count = 0;
    node temp(0, 0);
    que.push(temp); int time = 1;
    while (!que.empty())
    {
        node t = que.front();
```

```
        que.pop();

        if(t.value)
            ans[time - t.rank].push_back(t.value);

        count++;
        if (count >= 10)
        {
            count %= 10;
            time++;
        }

        for (int i = 0; i < adj_list[t.value].size(); ++i)
            que.push(node(t.rank + 1, adj_list[t.value][i]));
    }
}

int main()
{
    read_file();
    bfs();
    int sum = 0;
    for (int i = 0; i < 10; ++i)
    {
        if (ans[i].size() == 0)
            break;
        printf("时刻%d发送数据的节点为: \n", i);
        for (int j = 0; j < ans[i].size(); ++j)
            cout << ans[i][j] << ' ', sum++;
        cout << endl;
    }
    cout << "  xxxxx " << sum << endl;
    system("pause");
}
```

所需文件数据 adj_list.txt 如下：

```
0 10 42 43 47 48 67 68 72 73 74 98
42 2 22 37
43 3 18 38 49
47 3 17 41 46
48 2 23 44
67 2 62 97
68 1 63
72 3 66 71 92
73 2 69 93
```

```
74 7 45 50 70 75 94 95 100
98 7 99 117 118 119 122 123 124
17 1 12
22 1 16
37 1 32
41 1 36
46 1 21
18 1 13
23 1 19
38 2 33 39
49 1 20
62 1 57
66 1 61
71 1 91
92 1 112
97 2 96 121
63 1 58
69 1 64
93 1 88
45 1 40
50 2 24 25
70 1 65
94 1 90
95 2 114 115
99 1 120
100 1 125
117 1 87
118 1 113
119 1 89
122 1 116
12 1 7
16 1 11
32 1 27
36 1 31
13 1 8
19 1 14
33 1 28
39 1 35
20 1 15
40 1 34
57 2 52 82
61 1 56
91 1 111
112 1 107
```

116 1 86
58 1 53
64 1 59
88 1 83
65 1 60
113 1 108
89 1 110
90 1 84
114 1 109
115 1 85
7 1 2
11 1 6
27 1 26
8 1 3
14 1 9
15 1 10
28 1 29
35 1 30
52 1 77
56 1 51
86 1 81
107 1 102
111 1 106
53 2 54 78
83 1 103
60 1 55
84 1 105
85 1 104
109 1 80
110 1 79
2 1 1
3 1 4
29 1 5
51 1 76
77 1 101

==========题目三==========

代码一：

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <vector>
```

```
#include <queue>
using namespace std;

class node
{
public:
    int rank, value;
    node() = default;
    node(int _rank, int _value) :rank(_rank), value(_value) {}
};

vector<int> adj_list[130];

void read_file()
{
    ifstream file("adj_list.txt");
    int x, t, y;
    while (file >> x)
    {
        file >> t;
        for (int i = 0; i < t; ++i)
        {
            file >> y;
            adj_list[x].push_back(y);
        }
    }
}

vector<int> ans[10];



int calculate_size(int x)
{
    int count = 0;
    queue<int> que;
    que.push(x);
    while (!que.empty())
    {
        int t = que.front();
        que.pop();
        for (int i = 0; i < adj_list[t].size(); ++i)
                que.push(adj_list[t][i]);
        count++;
```

```
    }
    return count;
}

int main()
{
    read_file();
    int arr[] = { 42,43,47,48,67,68,72,73,74,98 };
    int brr[10];
    double x1 = 0.005;
    double x2 = 2001.35*0.000001;
    double y1 = 0.2;
    double y2 = 10004.5*0.00001;
    double time[10];
    for (int i = 0; i < 10; ++i)
        brr[i] = calculate_size(arr[i]);
    for (int i = 0; i < 10; ++i)
    {
        time[i] = 29.9/((x1 + x2 * brr[i]) / (y1 + y2 * brr[i]));
        cout << arr[i] << "可用    " << time[i] << endl;
    }

    system("pause");
}
```

9.1.2 MATLAB 代码

```
图 1
Y=ones(10);
h=bar3((0:9)+.5,Y,1);
set(h,'facecolor','w','linestyle',':')
for n=1:9
p=copyobj(h,gca);
for m=p'
set(m,'zdata',get(m,'zdata')+n);
end
end
grid off
axis equal
xlabel("x 轴")
ylabel("z 轴")
zlabel("y 轴")
```

图 5

```
x=1:1:51
y1=[125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,
125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,125,
125,125,125,125,125,125]
y2=[125,115,109,107,105,100,99,97,94,92,91,90,88,87,85,85,84,84,84,83,82,81,79,79,79,79,79,
79,79,79,79,79,79,79,79,79,79,79,79,79,79,79,79,79,79,79,79,79,79,79,79]
y3=[125,115,106,104,101,99,93,90,88,88,84,83,81,80,79,77,77,76,76,76,75,74,73,73,73,72,70,69,
69,69,69,69,68,68,67,67,67,67,67,66,66,66,65,65,65,65,64,64,64,64,64]
plot(x, y1)
hold on
plot(x, y2)
hold on
plot(x, y3)
xlabel("求解次数/次")
ylabel("传感器节点数量/个")
legend('100%','90%', '80%')
```

9.1.3 Unity 脚本代码

图 2 和图 3 的三维模型图来 unity 建模，其中 C#脚本代码如下：

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Test : MonoBehaviour
{
    public GameObject ball;
    public GameObject father;
    private float len = 0;
    private Vector3 beginPos;
    private bool flag = false;
    private float speed = 0.5f;
    private const float dis = 20;
    private int ball_nums;
    private void Awake()
    {
        ball_nums = 0;
        len = 40 / Mathf.Sqrt(3);
```

```
        beginPos = new Vector3(-50 + len / 2, -50 + len / 2, -50 + len / 2);
        //Debug.Log(beginPos);
    }

    private void Update()
    {
        if (beginPos.x - len/2 > 50)
        {
            father.transform.Rotate(Vector3.up * speed);
            return;
        }

        GameObject temp = Instantiate(ball, beginPos, Quaternion.identity);
        temp.transform.parent = father.transform;
        beginPos.z += len;
        if(beginPos.z - len/2 > 50)
        {
            beginPos.z = -50 + len / 2;
            beginPos.y += len;

            if(beginPos.y - len / 2> 50)
            {
                beginPos.y = -50 + len / 2;
                beginPos.x += len;
            }
        }
    }
}
```

9.2 数据

9.2.1 问题一数据

表 14 覆盖率为 100%时，节点数量及位置坐标表

	X	Y	Z		X	Y	Z		X	Y	Z		X	Y	Z		X	Y	Z
1	-38.453	-88.453	-38.453	26	-15.359	-88.453	-38.453	51	7.735	-88.453	-38.453	76	30.829	-88.453	-38.453	101	38.453	-88.453	-38.453
2	-38.453	-88.453	-15.359	27	-15.359	-88.453	-15.359	52	7.735	-88.453	-15.359	77	30.829	-88.453	-15.359	102	38.453	-88.453	-15.359
3	-38.453	-88.453	7.735	28	-15.359	-88.453	7.735	53	7.735	-88.453	7.735	78	30.829	-88.453	7.735	103	38.453	-88.453	7.735
4	-38.453	-88.453	30.829	29	-15.359	-88.453	30.829	54	7.735	-88.453	30.829	79	30.829	-88.453	30.829	104	38.453	-88.453	30.829
5	-38.453	-88.453	38.453	30	-15.359	-88.453	38.453	55	7.735	-88.453	38.453	80	30.829	-88.453	38.453	105	38.453	-88.453	38.453
6	-38.453	-65.359	-38.453	31	-15.359	-65.359	-38.453	56	7.735	-65.359	-38.453	81	30.829	-65.359	-38.453	106	38.453	-65.359	-38.453
7	-38.453	-65.359	-15.359	32	-15.359	-65.359	-15.359	57	7.735	-65.359	-15.359	82	30.829	-65.359	-15.359	107	38.453	-65.359	-15.359

8	-38.453	-65.359	7.735	33	-15.359	-65.359	7.735	58	7.735	-65.359	7.735	83	30.829	-65.359	7.735	108	38.453	-65.359	7.735
9	-38.453	-65.359	30.829	34	-15.359	-65.359	30.829	59	7.735	-65.359	30.829	84	30.829	-65.359	30.829	109	38.453	-65.359	30.829
10	-38.453	-65.359	38.453	35	-15.359	-65.359	38.453	60	7.735	-65.359	38.453	85	30.829	-65.359	38.453	110	38.453	-65.359	38.453
11	-38.453	-42.265	-38.453	36	-15.359	-42.265	-38.453	61	7.735	-42.265	-38.453	86	30.829	-42.265	-38.453	111	38.453	-42.265	-38.453
12	-38.453	-42.265	-15.359	37	-15.359	-42.265	-15.359	62	7.735	-42.265	-15.359	87	30.829	-42.265	-15.359	112	38.453	-42.265	-15.359
13	-38.453	-42.265	7.735	38	-15.359	-42.265	7.735	63	7.735	-42.265	7.735	88	30.829	-42.265	7.735	113	38.453	-42.265	7.735
14	-38.453	-42.265	30.829	39	-15.359	-42.265	30.829	64	7.735	-42.265	30.829	89	30.829	-42.265	30.829	114	38.453	-42.265	30.829
15	-38.453	-42.265	38.453	40	-15.359	-42.265	38.453	65	7.735	-42.265	38.453	90	30.829	-42.265	38.453	115	38.453	-42.265	38.453
16	-38.453	-19.171	-38.453	41	-15.359	-19.171	-38.453	66	7.735	-19.171	-38.453	91	30.829	-19.171	-38.453	116	38.453	-19.171	-38.453
17	-38.453	-19.171	-15.359	42	-15.359	-19.171	-15.359	67	7.735	-19.171	-15.359	92	30.829	-19.171	-15.359	117	38.453	-19.171	-15.359
18	-38.453	-19.171	7.735	43	-15.359	-19.171	7.735	68	7.735	-19.171	7.735	93	30.829	-19.171	7.735	118	38.453	-19.171	7.735
19	-38.453	-19.171	30.829	44	-15.359	-19.171	30.829	69	7.735	-19.171	30.829	94	30.829	-19.171	30.829	119	38.453	-19.171	30.829
20	-38.453	-19.171	38.453	45	-15.359	-19.171	38.453	70	7.735	-19.171	38.453	95	30.829	-19.171	38.453	120	38.453	-19.171	38.453
21	-38.453	-11.547	-38.453	46	-15.359	-11.547	-38.453	71	7.735	-11.547	-38.453	96	30.829	-11.547	-38.453	121	38.453	-11.547	-38.453
22	-38.453	-11.547	-15.359	47	-15.359	-11.547	-15.359	72	7.735	-11.547	-15.359	97	30.829	-11.547	-15.359	122	38.453	-11.547	-15.359
23	-38.453	-11.547	7.735	48	-15.359	-11.547	7.735	73	7.735	-11.547	7.735	98	30.829	-11.547	7.735	123	38.453	-11.547	7.735
24	-38.453	-11.547	30.829	49	-15.359	-11.547	30.829	74	7.735	-11.547	30.829	99	30.829	-11.547	30.829	124	38.453	-11.547	30.829
25	-38.453	-11.547	38.453	50	-15.359	-11.547	38.453	75	7.735	-11.547	38.453	100	30.829	-11.547	38.453	125	38.453	-11.547	38.453

表 15 覆盖率为 90%时，节点数量及位置坐标表

	X	Y	Z		X	Y	Z		X	Y	Z		X	Y	Z
1	-38.453	-91.453	-38.453	21	-27.359	-99.453	21.735	41	3.735	-88.453	5.735	61	17.829	-92.453	3.735
2	-19.453	-72.453	3.641	22	-34.359	-69.453	23.829	42	25.735	-88.453	23.829	62	27.829	-70.453	37.829
3	-32.453	-88.453	15.735	23	-2.359	-78.453	38.453	43	11.735	-88.453	42.923	63	37.829	-48.359	-35.453
4	-28.453	-82.453	49.829	24	-7.359	-74.359	-42.453	44	-10.265	-62.359	-29.453	64	19.829	-79.359	-14.359
5	-49.453	-75.359	-29.453	25	-24.359	-52.359	-9.359	45	15.735	-63.359	-7.359	65	20.829	-78.359	-0.265
6	-35.453	-46.359	-8.359	26	-29.359	-76.359	7.735	46	4.735	-77.359	15.735	66	46.829	-46.359	28.829
7	-38.453	-54.359	1.735	27	-0.359	-73.359	29.829	47	0.735	-65.359	40.829	67	44.829	-42.265	-19.453
8	-45.453	-62.359	34.829	28	-14.359	-56.359	38.453	48	7.735	-53.265	-44.453	68	25.829	-31.265	-29.359
9	-38.453	-23.265	-33.453	29	-1.359	-28.265	-38.453	49	8.735	-36.265	-0.359	69	13.829	-45.265	19.735
10	-23.453	-54.265	-21.359	30	-28.359	-38.265	-24.359	50	7.735	-31.265	13.735	70	30.829	-23.265	17.829
11	-39.453	-50.265	20.735	31	-6.359	-31.265	9.735	51	-9.265	-55.265	17.829	71	41.829	-29.171	-37.453
12	-31.453	-47.265	39.829	32	-12.359	-27.265	32.829	52	10.735	-53.265	44.923	72	21.829	-13.171	-15.359
13	-27.453	-57.265	38.453	33	0.641	-7.171	-44.453	53	26.735	-7.171	-40.453	73	48.829	-22.171	8.735
14	-38.453	-4.171	-37.453	34	-18.359	-18.171	-8.359	54	-4.265	-2.171	-34.359	74	31.829	-15.171	37.829
15	-38.453	-31.171	-4.359	35	-0.359	-5.171	10.735	55	-11.265	-13.171	10.735	75	49.923	-75.453	0.641
16	-45.453	-9.171	17.735	36	-10.359	-0.171	46.829	56	16.735	-32.171	45.829	76	38.453	-76.453	4.735
17	-19.453	-5.171	35.829	37	-25.359	-1.171	42.923	57	7.735	-20.171	49.923	77	38.453	-78.359	-47.453
18	-40.453	-5.077	-2.265	38	-8.359	-11.547	24.735	58	-0.265	-13.077	-31.453	78	36.923	-72.359	17.735
19	-30.359	-78.453	-45.453	39	5.735	-90.453	-24.453	59	35.829	-82.453	-40.453	79	41.923	-12.171	-8.265
20	-5.359	-78.453	-15.359	40	-10.265	-81.453	-13.359	60	30.829	-98.453	-26.359				

表 16 覆盖率为 90%时，节点数量及位置坐标表（局部最优解 1）

	X	Y	Z		X	Y	Z		X	Y	Z		X	Y	Z
1	-42.453	-80.453	-47.453	21	-28.359	-69.453	21.735	41	17.735	-70.453	19.735	61	29.829	-96.453	4.735
2	-34.453	-94.453	-13.359	22	-16.359	-103.453	40.829	42	7.735	-85.453	38.829	62	21.829	-91.453	21.829
3	-37.453	-88.453	20.735	23	-30.359	-77.453	42.923	43	4.735	-69.453	61.923	63	35.829	-75.359	-37.453
4	-24.453	-95.453	15.829	24	-20.359	-66.359	-35.453	44	-2.264	-62.359	-51.453	64	11.829	-54.359	-29.359
5	-44.453	-76.359	-49.453	25	0.641	-74.359	-22.359	45	4.735	-50.359	-11.359	65	14.829	-63.359	-6.26497
6	-35.453	-47.359	-28.359	26	-5.358	-73.359	11.735	46	24.735	-63.359	24.735	66	15.829	-81.359	14.829
7	-42.453	-55.359	-0.264	27	-12.359	-60.359	30.829	47	16.735	-51.359	42.829	67	41.829	-42.265	-24.453
8	-41.453	-79.359	25.829	28	-24.359	-69.359	64.923	48	-0.264	-33.265	-55.453	68	24.829	-29.265	-34.359
9	-46.453	-24.265	-36.453	29	-32.359	-46.265	-40.453	49	0.735	-48.265	-21.359	69	35.829	-55.265	6.735
10	-36.453	-34.265	-15.359	30	0.641	-57.265	-9.358	50	11.735	-44.265	16.735	70	24.829	-23.265	35.829
11	-27.453	-54.265	-3.264	31	-12.359	-50.265	16.735	51	-6.264	-51.265	37.829	71	33.829	-20.171	-57.453
12	-56.453	-30.265	35.829	32	-14.359	-30.265	43.829	52	-10.265	-32.265	45.923	72	49.829	-8.170	-18.359
13	-32.453	-44.265	40.923	33	-19.359	-38.171	-30.453	53	16.735	-27.171	-42.453	73	24.829	-21.171	16.735
14	-35.453	-27.171	-39.453	34	-11.359	-21.171	-27.359	54	8.735	-19.171	-32.359	74	45.829	-21.171	30.829
15	-27.453	-11.171	-5.358	35	-27.359	-24.171	5.735	55	18.735	-3.170	-10.265	75	39.923	-84.453	-27.359
16	-57.453	-26.171	19.735	36	-12.359	-30.171	42.829	56	4.735	-2.170	33.829	76	44.923	-76.453	1.735
17	-46.453	-8.170	33.829	37	-22.359	-4.170	54.923	57	5.735	-8.170	54.923	77	48.923	-58.359	-21.453
18	-32.453	-4.076	19.735	38	-4.358	-10.077	24.735	58	-1.264	-9.076	-46.453	78	43.923	-80.359	18.735
19	-34.359	-82.453	-24.453	39	10.735	-96.453	-45.453	59	23.829	-89.453	-34.453	79	43.923	-9.170	-10.265
20	-9.358	-99.453	-9.358	40	-11.265	-96.453	-17.359	60	39.829	-101.453	-29.359				

表 17 覆盖率为 90%时，节点数量及位置坐标表（局部最优解 2）

	X	Y	Z		X	Y	Z		X	Y	Z		X	Y	Z
1	-26.453	-72.453	-23.453	21	3.641	-85.453	-9.264	41	14.735	-100.453	19.735	61	30.829	-74.453	-3.264
2	-34.453	-94.453	-9.358	22	-3.358	-95.453	31.829	42	20.735	-95.453	19.829	62	34.829	-93.453	34.829
3	-21.453	-105.453	8.735	23	3.641	-77.453	34.923	43	2.735	-107.453	60.923	63	16.829	-79.359	-43.453
4	-21.453	-99.453	28.829	24	-22.359	-53.359	-39.453	44	14.735	-51.359	-36.453	64	40.829	-60.359	-3.358
5	-43.453	-54.359	-30.453	25	1.641	-74.359	-23.359	45	23.735	-82.359	-16.359	65	20.829	-54.359	-1.264
6	-41.453	-47.359	-22.359	26	-21.359	-65.359	7.735	46	4.735	-64.359	15.735	66	40.829	-65.359	12.829
7	-43.453	-78.359	-6.264	27	-8.358	-81.359	17.829	47	4.735	-61.359	35.829	67	46.829	-30.265	-45.453
8	-38.453	-71.359	33.829	28	-22.359	-46.359	43.923	48	11.735	-29.265	-21.453	68	40.829	-29.265	-24.359
9	-49.453	-43.265	-28.453	29	-25.359	-47.265	-31.453	49	21.735	-47.265	-11.359	69	30.829	-26.265	26.735
10	-42.453	-28.265	-2.358	30	-9.358	-42.265	-16.359	50	3.735	-38.265	20.735	70	35.829	-40.265	21.829

11	-29.453	-32.265	16.735	31	2.641	-39.265	-6.264	51	10.735	-46.265	15.829	71	32.829	-36.171	-52.453
12	-29.453	-43.265	17.829	32	-30.359	-45.265	28.829	52	4.735	-49.265	53.923	72	41.829	-8.170	-13.359
13	-42.453	-45.265	46.923	33	-10.359	-19.171	-44.453	53	15.735	-13.171	-37.453	73	30.829	-3.170	15.735
14	-21.453	-24.171	-37.453	34	-6.358	-8.177	-29.359	54	7.735	-21.171	-19.359	74	44.829	-9.170	28.829
15	-39.453	-7.170	-22.359	35	-26.359	-7.170	7.735	55	-6.264	-24.171	8.735	75	63.923	-69.453	-22.359
16	-51.453	-30.171	2.735	36	-15.359	-29.171	11.829	56	11.735	-9.170	17.829	76	35.923	-75.453	19.735
17	-36.453	-16.171	30.829	37	2.641	-2.170	41.923	57	0.735	-31.171	63.923	77	36.923	-52.359	-34.453
18	-27.453	-13.077	21.735	38	-26.359	-3.076	-5.264	58	-10.265	17.923	-32.453	78	52.923	-58.359	6.735
19	-19.359	-92.453	-44.453	39	6.735	-86.453	-37.453	59	49.829	-80.453	-52.453	79	62.923	-27.171	5.735
20	-28.359	-78.453	-23.359	40	-2.264	-98.453	-29.359	60	36.829	-100.453	-2.358				

表 18 覆盖率为 80%时，节点数量及位置坐标表

	X	Y	Z		X	Y	Z		X	Y	Z		X	Y	Z
1	-46.453	-83.453	-38.453	17	-11.359	-88.453	-23.453	33	21.735	-88.453	-19.453	49	34.829	-79.453	-22.453
2	-20.453	-92.453	-16.359	18	-15.359	-70.453	-32.359	34	-2.265	-88.453	-6.359	50	30.829	-88.453	-30.359
3	-48.453	-89.453	7.735	19	-3.359	-98.453	7.735	35	16.735	-70.453	7.735	51	24.829	-94.453	20.735
4	-34.453	-88.453	42.829	20	-19.359	-69.453	26.829	36	-0.265	-85.453	14.829	52	31.829	-91.453	31.829
5	-32.453	-61.359	-24.453	21	-25.359	-62.359	-34.453	37	21.735	-80.359	-42.453	53	39.829	-50.359	-38.453
6	-38.453	-56.359	-9.359	22	-11.359	-53.359	-4.359	38	7.735	-64.359	-27.359	54	32.829	-65.359	-32.359
7	-21.453	-46.359	19.735	23	-15.359	-65.359	0.735	39	-0.265	-80.359	18.735	55	48.829	-51.359	18.735
8	-23.453	-78.359	23.829	24	-0.359	-59.359	30.829	40	0.735	-81.359	44.829	56	24.829	-53.359	38.829
9	-38.453	-45.265	-35.453	25	-16.359	-36.265	-40.453	41	0.735	-37.265	-38.453	57	26.829	-29.265	-36.453
10	-29.453	-29.265	-5.359	26	-11.359	-38.265	-12.359	42	26.735	-34.265	-26.359	58	39.829	-48.265	-33.359
11	-38.453	-57.265	3.735	27	-17.359	-53.265	12.735	43	6.735	-40.265	16.735	59	30.829	-25.265	17.735
12	-38.453	-54.265	44.829	28	-21.359	-34.265	37.829	44	8.735	-27.265	23.829	60	31.829	-42.265	46.829
13	-35.453	-19.171	-45.453	29	-15.359	-15.171	-21.453	45	2.735	-14.171	-43.453	61	39.829	-5.171	-47.453
14	-38.453	-30.171	-16.359	30	2.641	-0.171	3.641	46	20.735	-3.171	-33.359	62	32.829	-23.171	-5.359
15	-19.453	-5.171	-8.265	31	-16.359	-26.171	14.735	47	-10.265	-5.171	-1.265	63	17.829	-30.171	12.735
16	-24.453	-37.171	42.829	32	-32.359	-2.171	30.829	48	4.735	-24.171	41.829	64	24.829	-8.171	20.829

表 19 覆盖率为 80%时，节点数量及位置坐标表（局部最优解 1）

	X	Y	Z		X	Y	Z		X	Y	Z		X	Y	Z
1	-20.453	-94.453	-49.453	17	1.641	-102.453	-38.453	33	12.735	-76.453	-49.453	49	27.829	-74.453	-19.453
2	-38.453	-104.453	-15.359	18	-30.359	-103.453	1.641	34	15.735	-75.453	2.641	50	34.829	-98.453	-5.358
3	-47.453	-106.453	8.735	19	-13.359	-99.453	-3.264	35	-0.264	-91.453	2.735	51	30.829	-80.453	-11.265
4	-29.453	-80.453	33.829	20	-32.359	-84.453	13.829	36	24.735	-91.453	25.829	52	25.829	-84.453	35.829
5	-35.453	-48.359	-25.453	21	-31.359	-73.359	-24.453	37	15.735	-83.359	-45.453	53	39.829	-78.359	-19.453
6	-56.453	-79.359	-22.359	22	-11.359	-78.359	-23.359	38	16.735	-74.359	-19.359	54	49.829	-46.359	-30.359

7	-57.453	-52.359	-5.264	23	-24.359	-60.359	25.735	39	21.735	-68.359	-7.264	55	21.829	-53.359	8.735
8	-20.453	-70.359	28.829	24	-34.359	-58.359	30.829	40	2.735	-55.359	44.829	56	44.829	-82.359	46.829
9	-57.453	-51.265	-41.453	25	-15.359	-46.265	-36.453	41	7.735	-55.265	-26.453	57	12.829	-32.265	-55.453
10	-41.453	-32.265	-34.359	26	3.641	-26.265	-4.358	42	22.735	-41.265	-20.359	58	21.829	-31.265	-32.359
11	-35.453	-30.265	6.735	27	-10.359	-28.265	24.735	43	6.73503	-51.265	-1.264	59	44.829	-50.265	23.735
12	-23.453	-48.265	35.829	28	-30.359	-59.265	16.829	44	16.735	-59.265	18.829	60	43.829	-26.265	19.829
13	-42.453	-12.171	-32.453	29	-32.359	-15.171	-41.453	45	19.735	-22.171	-35.453	61	37.829	-20.171	-21.453
14	-56.453	-22.171	-15.359	30	-20.359	-9.170	-30.359	46	-2.264	-24.171	-31.359	62	41.829	-6.170	-9.358
15	-34.453	-35.171	4.735	31	-18.359	-6.170	4.735	47	11.735	-32.171	-2.264	63	30.829	-13.171	22.735
16	-37.453	-33.171	35.829	32	-20.359	-17.171	32.829	48	-10.265	-3.170	30.829	64	29.829	-14.171	45.829

表 20 覆盖率为 80%时，节点数量及位置坐标表（局部最优解 2）

	X	Y	Z		X	Y	Z		X	Y	Z		X	Y	Z
1	-50.453	-78.453	-50.453	17	-11.359	-77.453	-27.453	33	23.735	-92.453	-24.453	49	35.829	-99.453	-31.453
2	-49.453	-77.453	-30.359	18	-26.359	-83.453	0.641	34	15.735	-71.453	-14.359	50	43.829	-72.453	-20.359
3	-48.453	-69.453	-0.264	19	-4.358	-96.453	1.735	35	1.735	-75.453	2.735	51	31.829	-78.453	8.735
4	-54.453	-86.453	11.829	20	2.642	-75.453	21.829	36	-7.264	-78.453	11.829	52	34.829	-107.453	39.829
5	-52.453	-59.359	-32.453	21	-32.359	-73.359	-39.453	37	6.735	-55.359	-47.453	53	25.829	-65.359	-20.453
6	-44.453	-49.359	-3.358	22	-34.359	-51.359	-0.358	38	10.735	-60.359	-2.358	54	30.829	-48.359	2.641
7	-45.453	-56.359	7.735	23	-6.358	-61.359	15.735	39	8.735	-68.359	-4.264	55	34.829	-49.359	-3.264
8	-28.453	-83.359	32.829	24	-14.359	-58.359	35.829	40	14.735	-61.359	34.829	56	30.829	-70.359	39.829
9	-36.453	-44.265	-54.453	25	0.641	-33.265	-28.453	41	18.735	-55.265	-54.453	57	22.829	-26.265	-30.453
10	-36.453	-29.265	-31.359	26	1.641	-40.265	-24.359	42	-5.264	-53.265	-9.358	58	46.829	-34.265	-19.359
11	-43.453	-57.265	24.735	27	3.641	-51.265	-6.264	43	7.735	-53.265	-9.264	59	21.829	-47.265	10.735
12	-36.453	-45.265	39.829	28	-4.358	-42.265	44.829	44	1.735	-26.265	14.829	60	47.829	-50.265	30.829
13	-32.453	-10.171	-30.453	29	-8.358	-24.171	-51.453	45	26.735	-29.171	-22.453	61	27.829	-7.170	-43.453
14	-55.453	-3.170	-23.359	30	-30.359	-25.171	-31.359	46	5.735	-21.171	-5.358	62	47.829	-19.171	-19.359
15	-55.453	-36.171	-10.265	31	-16.359	-19.171	3.735	47	6.735	-4.170	8.735	63	41.829	-4.170	22.735
16	-30.453	-19.171	48.829	32	-33.359	-16.171	20.829	48	19.735	-17.171	21.829	64	39.829	-22.171	42.829

9.2.2 问题二数据

9.2.3 问题三数据